E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Modelado 3D de cabeza mediante Kinect

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Andoni Larumbe Bergera

Arantzazu Villanueva Larre

Rafael Cabeza Laguna

Pamplona, 30 de Junio de 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen

Este proyecto se engloba en un marco más amplio consistente en el desarrollo de técnicas de seguimiento de la mirada empleando hardware low cost. Una de las partes más críticas de esta tarea consiste en la determinación precisa de la posición de la cabeza en 3D. Los algoritmos de estimación 3D propuestos están basados en puntos 2D obtenidos de la imagen y en un modelo 3D del objeto a seguir. Es en esta última parte en la que se engloban las tareas asociadas a este proyecto. El objetivo principal consiste en validar Kinect v2 (periférico de Microsoft) como herramienta para la construcción de modelos 3D de la cabeza. La tarea consiste en, mediante el uso de Kinect v2, obtener datos de más de 1000 puntos 3D de la cabeza y, posteriormente, usarlos para realizar un modelado virtual de la misma a través de un análisis PCA (Principal Component Analysis) de un modelo "estándar" BFM (Basel Face Model). El modelo estándar BFM se deformará de manera controlada para lograr una reconstrucción lo más fiel posible al modelo obtenido mediante Kinect.

Los resultados servirán para compararlos con los obtenidos empleando otros modelos y con los obtenidos en proyectos anteriores.

Abstract

This project is included in the scope of the development of eye tracking techniques involving low-cost hardware. One of the most critical tasks during this process is precisely determining of the head's three-dimensional position. The proposed 3D estimation algorithms are based on 2D points obtained from an image and a 3D model of the object being tracked. It's at this point that the tasks associated to this project take place. The main objective of this project is to validate Kinect v2 (Microsoft peripheral device) as a tool for the construction of a head's 3D model. The task involves being able to obtain data of more than 1000 3D points of the head to use them to elaborate a virtual model of the user through a PCA analysis (Principal Component Analysis) of a standard BFM (Basel Face Model) model. The BFM will then be shaped in a controlled way to achieve a reconstruction as faithful as possible to the model obtained with Kinect.

The results will be compared with those obtained using different models and those of previous projects.

Lista de palabras clave

Kinect v2

Modelo deformable 3D

Nube de puntos

Basel Face Model

Componentes principales

ICP

Procrustes

Índice de contenido

1. **Introducción** **2**
2. **Objetivos** **4**
3. **Estado del arte** **6**
4. **Hardware** **8**
   4.1 Kinect v2 8
      4.1.1 Aspectos generales 8
      4.1.2 Montaje y colocación 10
      4.1.3 Requisitos del sistema 13
   4.2 Cabeza sintética: Dasha 13
5. **Software** **14**
   5.1 MATLAB 14
   5.2 Visual Studio 16
   5.3 C# 18
   5.4 SDK Kinect 18
   5.5 Basel Face Model 21
6. **Procedimiento y Resultados** **24**
   6.1 Dasha 25
      6.1.1 Tratamiento de la nube de puntos de Dasha 25
      6.1.2 Ajuste del modelo BFM con la nube de puntos de Dasha 28
   6.2 Programación Kinect 34
   6.3 Tratamiento previo a la nube de puntos de Kinect 36
   6.4 Ajuste del modelo 40
7. **Conclusiones y líneas futuras** **48**
8. **Manual de usuario** **50**
   8.1 Obtención de la nube de puntos de Kinect 50
   8.2 Programa "*showFaces*" 54
      8.2.1 Introducción 54
      8.2.2 Manual de usuario 55
9. **Bibliografía** **58**

Capítulo 1
Introducción

El grupo de investigación biomédica de la Universidad Pública de Navarra lleva años trabajando en proyectos relacionados con el seguimiento de la mirada (en inglés, "*eye-tracking*"). Actualmente, miembros de este grupo de investigación están trabajando en un proyecto para la: "Interacción ubicua basada en la mirada para dispositivos móviles". Este proyecto tiene como finalidad desarrollar una tecnología que permita la utilización del eye-tracking en los dispositivos móviles.

El eye-tracking es el proceso por el cual estimamos la dirección de la mirada de un sujeto frente a una escena. Este proceso puede ser utilizado para numerosas aplicaciones de una variedad de disciplinas muy dispares. Un claro ejemplo de una disciplina que se puede aprovechar del eye-tracking es el de la publicidad; mediante el eye-tracking se puede saber hacia donde dirige su mirada un usuario al observar un anuncio, un producto o una página web. A partir de estos datos se realizaría un estudio de marketing para una correcta distribución de la publicidad en las páginas web o una correcta colocación de los productos en los centros comerciales.

Otra importante aplicación sería la de sustituir el ratón por un programa que detectase el punto de la pantalla al que estamos mirando y; mediante alguna acción específica como el parpadear rápidamente o el guiñar un ojo, se pudiese sustituir el clic del ratón. Esta aplicación sería muy interesante para eliminar algunas de las barreras fisiológicas que tienen algunas personas discapacitadas frente al mundo de los ordenadores o para facilitar la labor de algunas personas que, por diversos motivos, necesitan el uso de algún programa y no pueden hacerlo por si mismos porque tienen las manos ocupadas como puede ser el caso de los cirujanos en mitad de una operación. También se podrían realizar estudios acerca de las capacidades cognitivas de nuestro cerebro, desarrollar sistemas que permitiesen saber si un usuario está fatigado, mejorar los algoritmos de compresión basándose en las zonas en las que el usuario presta menos atención etc.

Una parte crítica a la hora de realizar el eye-tracking es saber exactamente donde se encuentran los ojos del usuario. Para ello es necesario tener un buen modelado de la cabeza del usuario y saber en qué posición se encuentra. Es aquí donde reside el sentido de este proyecto; en realizar un modelado 3D de la cabeza del usuario mediante Kinect y determinar si este dispositivo resulta válido para ello.

Una vez realizado este modelado, podría exportarse a cualquier dispositivo para que, mediante un login o una simple detección facial, se relacionase con el usuario al que pertenece; facilitando así la computación en dicho dispositivo.

Capítulo 2
Objetivos

Podemos dividir el proyecto en varias fases, cada una de ellas con sus propios objetivos.

En una primera fase, se trabajará con la nube de puntos extraída a partir de una cabeza sintética. El objetivo principal de esta fase será el de familiarizarse con los conceptos relacionados con los modelos 3D; como pueden ser los distintos tipos de rotaciones 3D (ver figura 2.1), las matrices de rotación-traslación o incluso los cuaterniones.

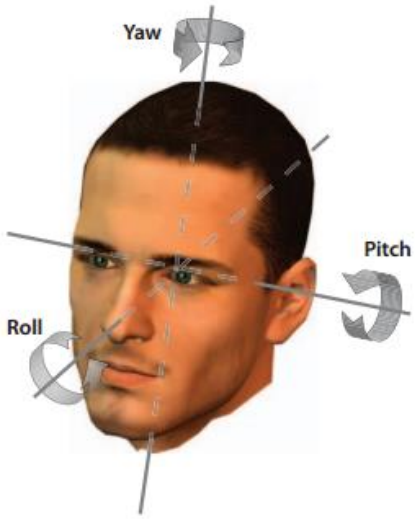

Fig.2.1 *Tipos de rotaciones 3D* **[1]**

Este proyecto también tiene una parte importante de programación. Por lo que otro objetivo será el de la familiarización con los lenguajes de programación a utilizar así como con los entornos en los que se realizarán tanto los programas como los experimentos.

En otra fase, se necesitará el uso de conceptos y algoritmos más complejos para el tratamiento tanto de nubes de puntos como de modelos deformables. Algunos de estos algoritmos (que ya detallaremos en capítulos posteriores) son el PCA (Principal Component Analysis), el ICP (Iterative Closest Point) o la triangulación (ver figura 2.2).

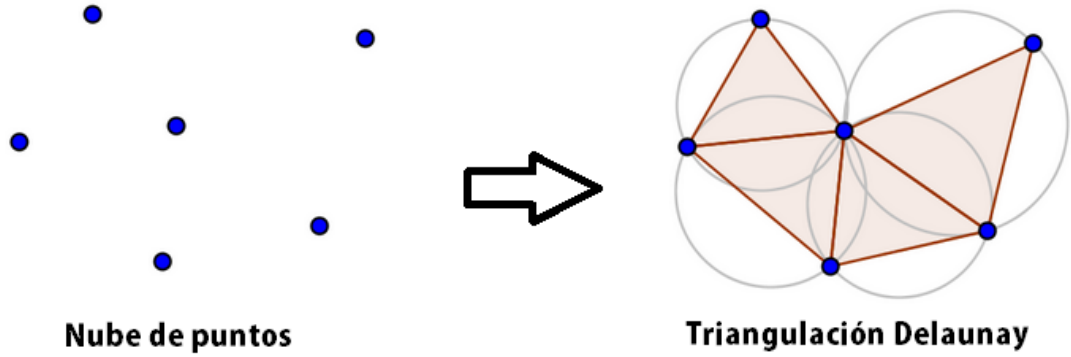

Fig. 2.2 *Ejemplo de triangulación de una nube de puntos* **[2]**

Por último, es necesario abordar la parte más científica de diseño de los experimentos y validación de los resultados. El objetivo de esta parte será el de la evaluación de los resultados y las conclusiones en las que se valorará la utilidad de la herramienta Kinect v2 como método para la modelización 3D de la cabeza.

Capítulo 3
Estado del arte

Los algoritmos de Structure-from-Motion (SfM) son capaces de estimar la estructura de un objeto 3D a partir de imágenes 2D del mismo. El uso de estos algoritmos se ha extendido desde la década de los 90; década en la que Tomasi y Kanade **[3]** propusieron un algoritmo capaz de estimar la forma de un cuerpo rígido mediante el uso de una cámara ortográfica. Este algoritmo fue mejorando hasta conseguir que fuese capaz de estimar la forma de objetos no rígidos mediante el uso de una gran variedad de tipos de cámaras.

Por otra parte, el uso de algoritmos de modelos de apariencia parametrizados (**PAMs**, del inglés, "***P**arameterized **A**ppearance **M**odels*") y de modelos de apariencia activa (**AAMs** del inglés, "***A**ctive **A**ppearance **M**odels*") como herramientas para modelar las formas y aspectos de los rostros, se ha extendiendo en los últimos años. Esto se debe a que estos algoritmos tienen numerosas ventajas frente a otros, pero aun así, tienen algunos inconvenientes como por ejemplo que son muy propensos a generar mínimos locales en el proceso de adaptación. Además, muchos de estos mínimos locales no corresponden a soluciones aceptables. Otro problema es su mala respuesta ante la detección de las identidades de los sujetos, de movimientos 3D y de expresiones **[4]**.

El uso de modelos deformables 3D (**3DMMs**, del inglés, "*3D **M**orphable **M**odels*") soluciona algunos de estos problemas. Los modelos creados mediante estos algoritmos están capacitados para distinguir la identidad y las expresiones de un sujeto aunque éste cambie de pose. Estas y otras características hacen que el uso de estos modelos deformables sea prometedor, aunque existen algunos inconvenientes:

- El número de bases de datos 3D es bastante limitado y, generalmente, no hay mucha variedad en las expresiones faciales.
- La mayoría de los modelos 3D se forman a través de escaneos láser, lo que requiere complejas configuraciones. Además, en la mayoría de los casos, es necesario que los sujetos tengan una expresión neutral.
- Encontrar correspondencias y generar el modelo 3D es un proceso laborioso y muy propenso a errores.

En nuestro caso y como ya se detallará en capítulos posteriores, haremos uso de un modelo deformable 3D, obtenido de una base de datos de la Universidad de Besel (Suiza), para generar modelos de diferentes caras. Utilizaremos también una nube de puntos 3D para deformar el modelo y ver si el resultado obtenido es viable. Esta nube de puntos será generada por un dispositivo de Microsoft mediante la técnica SfM.

Capítulo 4
Hardware

En este capítulo se detallarán los diferentes dispositivos u objetos de los que vamos a hacer uso directa o indirectamente en este proyecto.

4.1 Kinect v2

4.1.1 Aspectos generales

Para extraer la nube de puntos que, más tarde, usaremos para generar el modelo 3D de la cabeza, utilizaremos la segunda y mejorada versión del periférico de Microsoft: Kinect.

Este dispositivo es capaz de percibir imágenes 1080p HD, imágenes infrarrojas, profundidad de los elementos de la imagen, origen del sonido… todo esto y más gracias a la gran cantidad de sensores de los que dispone. Estos sensores son:

- **Cámara RGB:** resolución de 1920x1080 capaz de tomar 30 o 15 fps (fotogramas por segundo) dependiendo del nivel de luz.
- **Emisores IR (infrarrojos):** resolución de 512x424 capaz de tomar valores de intensidad de 16 bits a 30 fps.
- **Sensor de profundidad:** capaz de calcular, gracias a los emisores IR, las distancias (en milímetros) respecto al sensor con valores de 16 bits en un rango de 0.5 a 8 metros.
- **Array de micrófonos.**

En la figura 4.1 podemos ver la misma imagen tomada por los sensores de color RGB, IR y profundidad.

Fig 4.1 *Misma escena tomada por los distintos sensores de Kinect* **[5]**

En la figura 4.2 podemos ver el dispositivo Kinect v2 y la localización aproximada de los sensores comentados anteriormente.

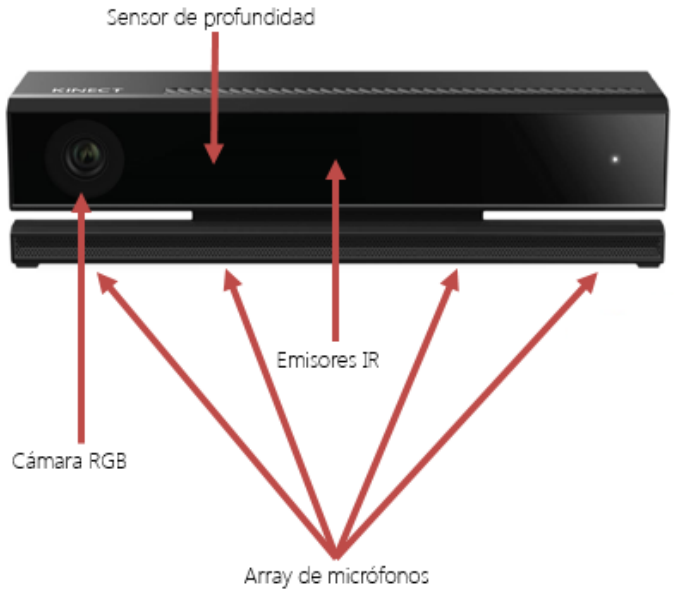

Fig. 4.2 *Dispositivo Kinect v2* **[5]**

En la figura 4.3 se puede ver en mas detalle la localización de los sensores de profundidad, RGB e infrarrojos.

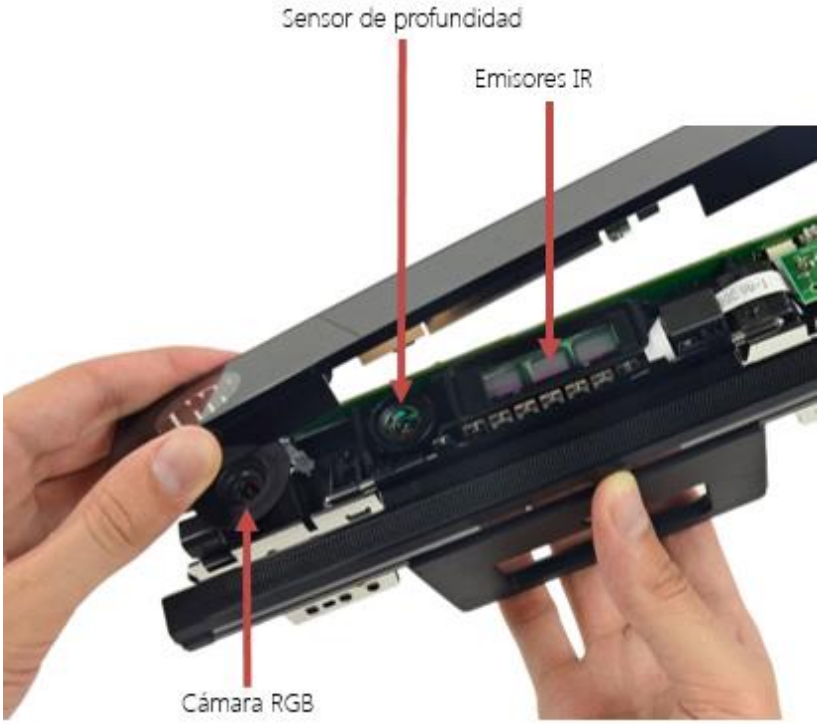

Fig. 4.3 *Detalle de los sensores de profundidad, RGB e infrarrojos* **[5]**

El periférico de Microsoft está diseñado para funcionar tanto en Windows como en Xbox One (la videoconsola de última generación de Microsoft) pero nosotros nos centraremos en su uso para Windows.

Para su uso en PC es necesaria la utilización de un adaptador que dote al dispositivo de corriente y que permita la salida de los datos en el estándar de conexión USB 3.0 tal y como se ve en la figura 4.4.

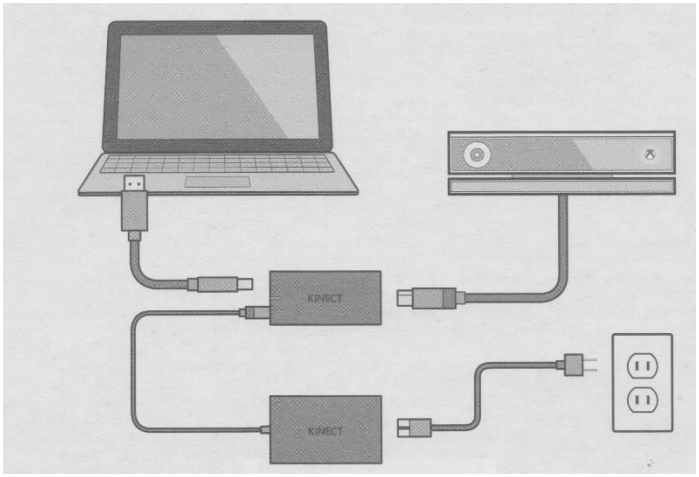

Fig 4.4 *Esquema de conexión*

4.1.2 Montaje y colocación

Como primer paso conectaremos la salida de datos del dispositivo Kinect a la entrada del adaptador para PC como se ve en la figura 4.5.

Fig. 4.5 *Conexión Kinect-adaptador*

A su vez conectaremos el adaptador al transformador de corriente (figura 4.6) y éste a la toma eléctrica (figura 4.7).

Fig. 4.6 *Conexión adaptador-transformador*

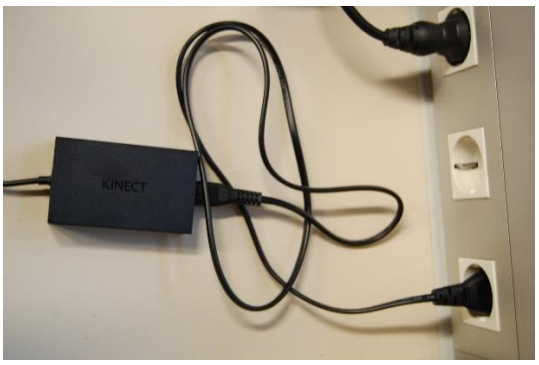

Fig. 4.7 *Conexión transformador-toma eléctrica*

Por último conectaremos la salida del adaptador a un puerto USB 3.0 de nuestro PC tal y como se ve en la figura 4.8.

Fig. 4.8 *Conexión adaptador-puerto USB*

Para la correcta sujeción de Kinect utilizaremos un trípode tal y como se ve en la figura 4.9.

El dispositivo Kinect podrá colocarse en la posición y la altura que se desee pero es recomendable ajustarlo en una posición tal, que se encuentre frente al sujeto del que se quieren tomar los datos y a una altura superior al monitor del PC al que está conectado (figura 4.10).

Fig. 4.9 *Colocación del trípode*

Fig. 4.10 *Posición recomendada*

4.1.3 Requisitos del sistema

Dada la gran cantidad de datos con los que trabaja Kinect (datos de fotogramas RGB, IR, datos de profundidad…) se necesita una altísima velocidad de transferencia y procesado de los mismos. Por ello es obligatorio que nuestro PC cumpla los siguientes requisitos:

- **Sistema Operativo**: Windows 8.0/8.1 (x64).
- **CPU**: I7 3.1 GHz o superior.
- **RAM**: 4GB o superior.
- **GPU:** Adaptador de gráficos compatible con DirectX 11.
- **USB:** Versión 3.0. El circuito integrado deberá soportar Gen-2 y ser Intel o Renesas compatible con Windows 8.

4.2 Cabeza sintética: Dasha

Por otro lado, es necesario para el completo entendimiento de la primera parte del proyecto explicar quién, o mejor dicho, qué es Dasha.

Dasha es el nombre que se le da a una determinada cabeza sintética en el Laboratorio de Proyectos, Teoría de la señal y Comunicaciones. A partir de esta cabeza, se obtuvo la nube de puntos con la que comenzamos a trabajar en la primera parte de este proyecto pero de esto ya se hablará en los siguientes capítulos.

En las figuras 4.11 y 4.12 podemos ver dos fotografías de la cabeza denominada Dasha.

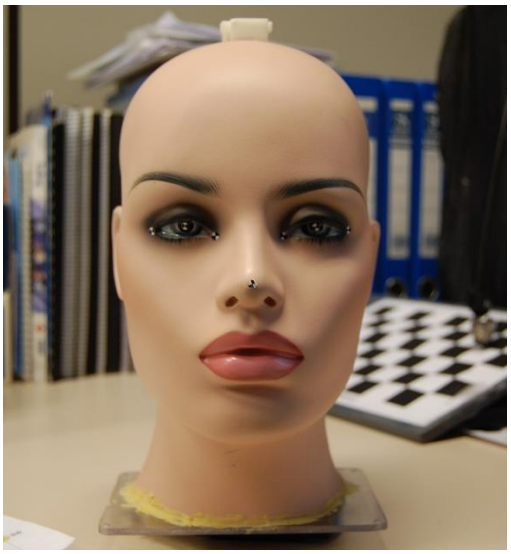

Fig. 4.11 *Dasha*

Fig. 4.12 *Perfil de Dasha*

Capítulo 5
Software

En este capítulo se detallarán los diferentes entornos de trabajo utilizados así como los lenguajes de programación necesarios para el desarrollo de este proyecto. También se detallarán las bases de datos utilizadas para generar los modelados de cabeza 3D.

5.1 MATLAB

MATLAB es un lenguaje de programación de alto nivel enfocado en los cálculos matemáticos y, en particular, en los matriciales. Tiene un entorno de desarrollo propio el cual podemos ver en la figura 5.1.

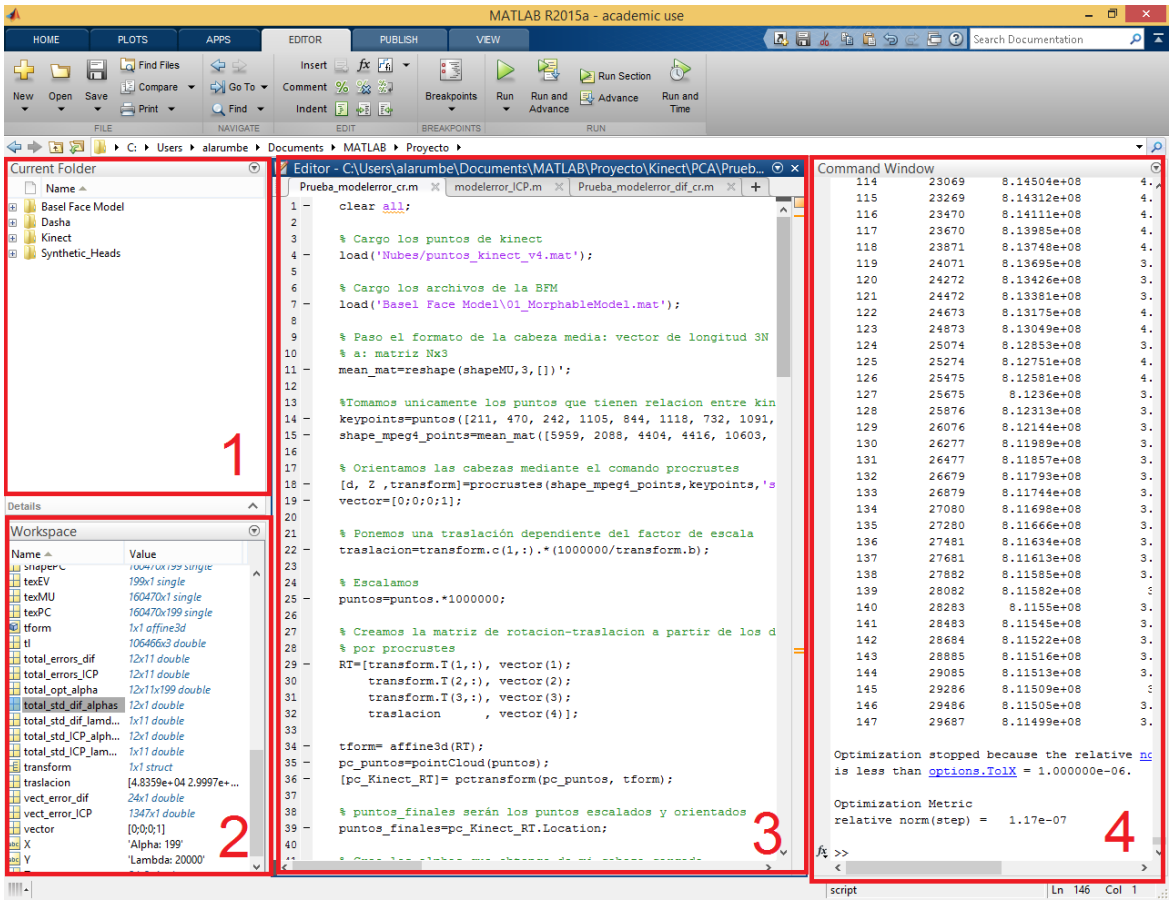

Fig. 5.1 *Entorno de desarrollo MATLAB R2015a*

Como se puede observar en la figura, nosotros usaremos la versión MATLAB R2015a. Esto es importante debido a que esta versión dispone de ciertas funciones necesarias para el correcto funcionamiento de nuestro programa. También se puede observar en la figura que el entorno de MATLAB dispone de distintas ventanas de trabajo:

- **La ventana 1** (ver figura) es la que determina en que carpeta estamos trabajando. Aquí podremos ver las diferentes subcarpetas, ficheros, funciones… que hayamos creado. Es muy útil para mantener ordenados nuestros scripts.
- **La ventana 2** es el denominado "Workspace". Aquí se guardarán las variables que estemos utilizando en un determinado momento.
- **La ventana 3** es el editor. Aquí escribiremos los programas y funciones que más tarde ejecutaremos. Es imprescindible para escribir programas largos y para detectar errores en ellos.
- **La ventana 4** es la ventana de comandos. Aquí ejecutaremos los programas y funciones creadas mediante el editor. También se pueden ejecutar sencillas líneas de comandos para testear funciones, crear variables…

Será en este entorno donde se realizará la mayor parte del trabajo, siendo aquí donde se trate a la nube de puntos proveniente de Kinect para adecuarla al modelo BFM (Basel Face Model) del que más tarde hablaremos. También será aquí donde se realicen las pruebas para determinar los valores óptimos de los diferentes parámetros que se encargan de generar nuestro modelado 3D.

Como ya hemos comentado MATLAB es un entorno orientado a los cálculos numéricos por lo que es el ideal para realizar nuestros análisis de error y generar las gráficas que nos proporcionarán punto de vista más visual de los resultados obtenidos.

Por otro lado MATLAB dispone de una interfaz de programación gráfica denominada GUIDE (Graphical User Interface Development Enviroment) al que se accede desde la pestaña HOME tal y como se muestra en la figura 5.2. Mediante esta interfaz, y como ya se detallará en capítulos posteriores, crearemos una simple aplicación capaz de mostrar las diferentes cabezas generadas mediante la selección de dos parámetros.

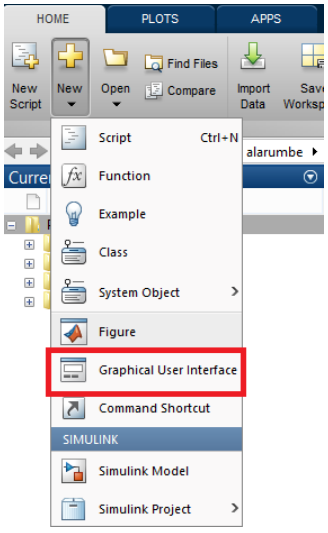

Fig. 5.2 *Acceso al GUIDE de MATLAB*

5.2 Visual Studio

Visual Studio es un entorno de desarrollo específico para sistemas operativos Windows. Este entorno nos permite crear aplicaciones para Windows, Android e iOS además de aplicaciones web y servicios de "nube" en cualquier entorno que soporte la plataforma .NET. Así pues, se pueden crear aplicaciones que se comuniquen entre dispositivos móviles, consolas, páginas web, estaciones de trabajo…

Por otro lado soporta múltiples lenguajes de programación como pueden ser Visual Basic .NET, Python, Ruby, PHP, C++, C# o Java lo que le dota de una gran flexibilidad y permite que un mayor número de desarrolladores pueda usarlo como herramienta para crear sus aplicaciones.

Para este proyecto utilizaremos la versión Visual Studio 2013. Podemos ver el entorno de desarrollo de esta versión en la figura 5.3.

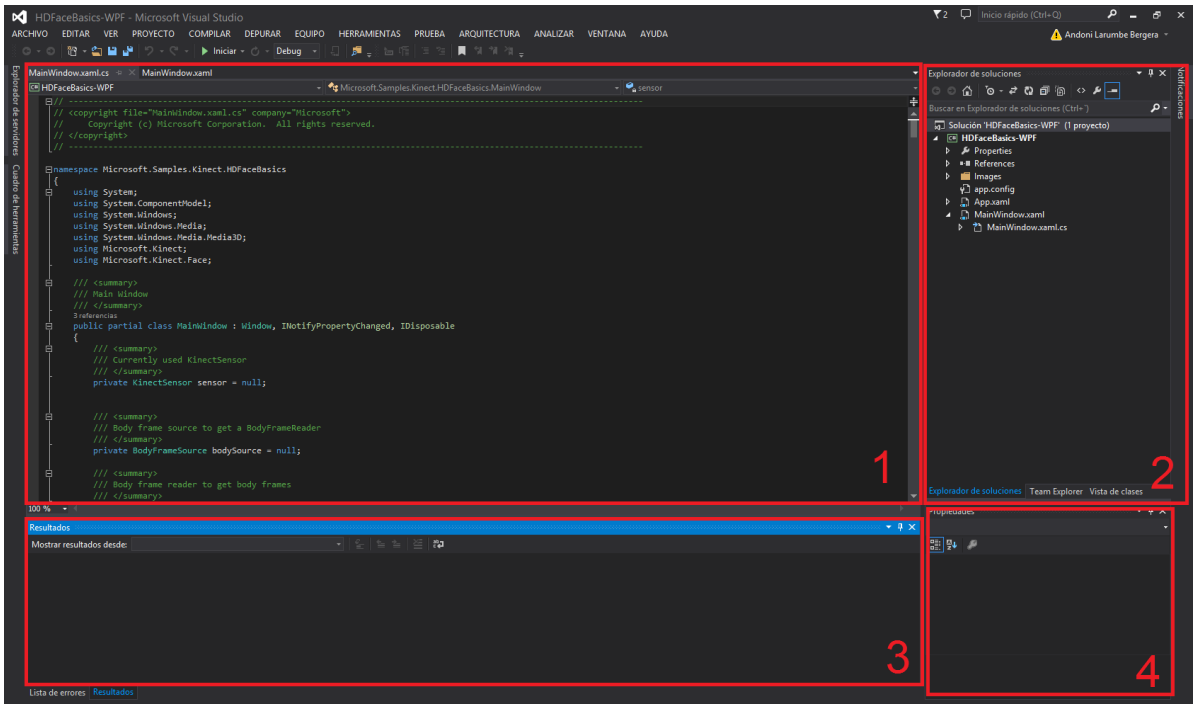

Fig. 5.3 *Entorno de desarrollo Visual Studio 2013*

Al igual que ocurre en MATLAB R2015a, el entorno desarrollo de Visual Studio 2013 también dispone de diferentes ventanas (ver figura 5.3):

- **La ventana 1** es el editor de código. Aquí escribiremos los programas que más tarde ejecutaremos.
- **La ventana 2** es el explorador de soluciones. Nos proporciona una vista organizada de los proyectos y sus archivos.

- **La ventana 3** es la ventada de resultados. Aquí se muestran los mensajes de estado de diversas características del entorno de desarrollo.
- **La ventana 4** es la ventana de propiedades. Aquí se pueden ver y cambiar las propiedades de los archivos y proyectos.

Visual Studio también dispone de un entorno de desarrollo gráfico (ver figura 5.4) del cual haremos uso en este proyecto. Las ventanas de este entorno son muy similares a las del anteriormente mostrado:

- **La ventana 1** nos permite diseñar gráficamente nuestra aplicación. Mediante el uso del ratón podemos agregar elementos, moverlos, aumentarlos…
- **Las ventanas 2, 3 y 4** son equivalentes a las del apartado anterior.
- **La ventana 5** permite agregar texto y elementos para, a continuación, colocarlos, ajustar su tamaño y configurar las propiedades mediante menús especiales. Cabe destacar que, al contrario que en la ventana 1, en esta ventana solo se puede programar mediante el uso de código escrito.

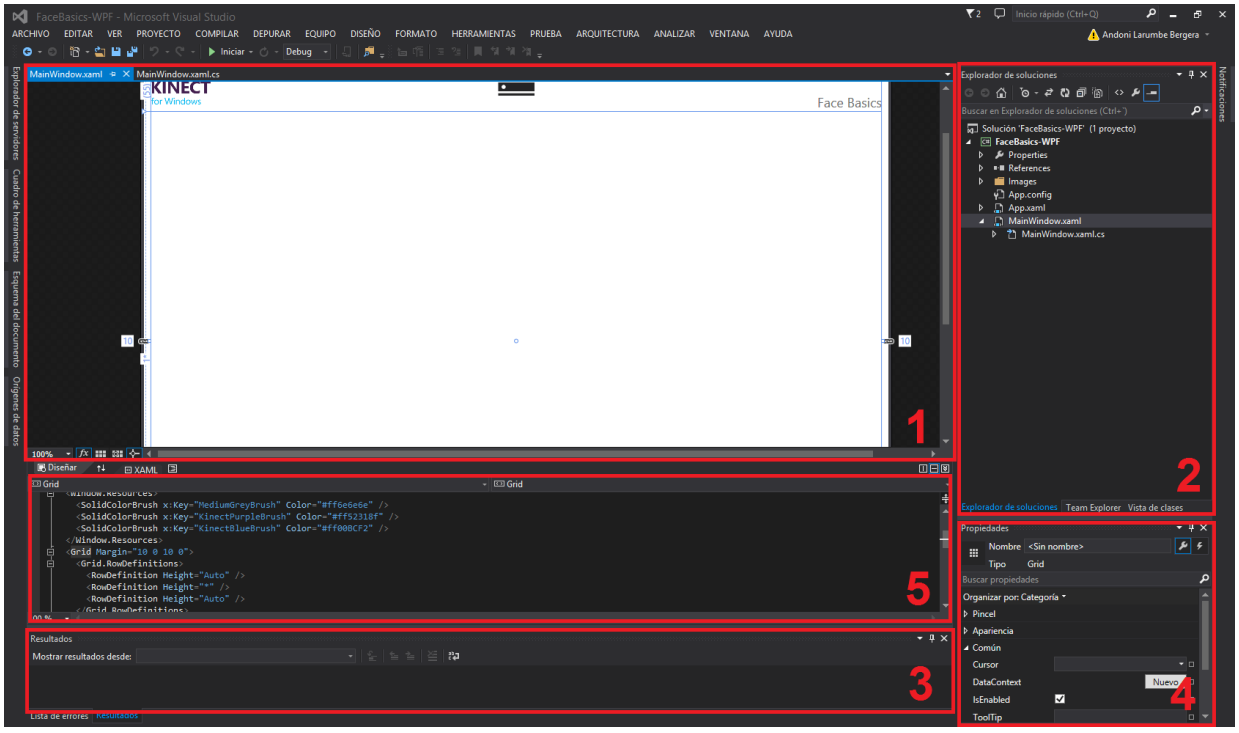

Fig. 5.4 *Entorno de desarrollo gráfico Visual Studio 2013*

Utilizaremos Visual Studio como el entorno sobre el que realizaremos las modificaciones a las aplicaciones tomadas de la SDK de Kinect.

5.3 C#

C# (C Sharp) es un lenguaje de programación de alto nivel orientado a objetos. Fue desarrollado y estandarizado por Microsoft como parte de su plataforma .NET y, como ya hemos dicho, está soportado por el entorno de desarrollo Visual Basic 2013.

Utilizaremos C# como el lenguaje mediante el cual realizaremos las modificaciones a las aplicaciones tomadas de la SDK de Kinect.

5.4 SDK Kinect

Un Kit de desarrollo de software, también llamado SDK (del inglés, Software Development Kit) es un conjunto de herramientas y/o programas de desarrollo de software que permiten al programador crear aplicaciones para un determinado paquete de software, plataforma de hardware, sistema operativo, videoconsola…

En nuestro caso, el SDK de Kinect nos proporciona además el programa "SDK Browser" para, entre otras muchas cosas, la descarga de documentación y códigos de ejemplo que nos servirán para el desarrollo de aplicaciones básicas. Podemos ver una imagen de este programa en la figura 5.5.

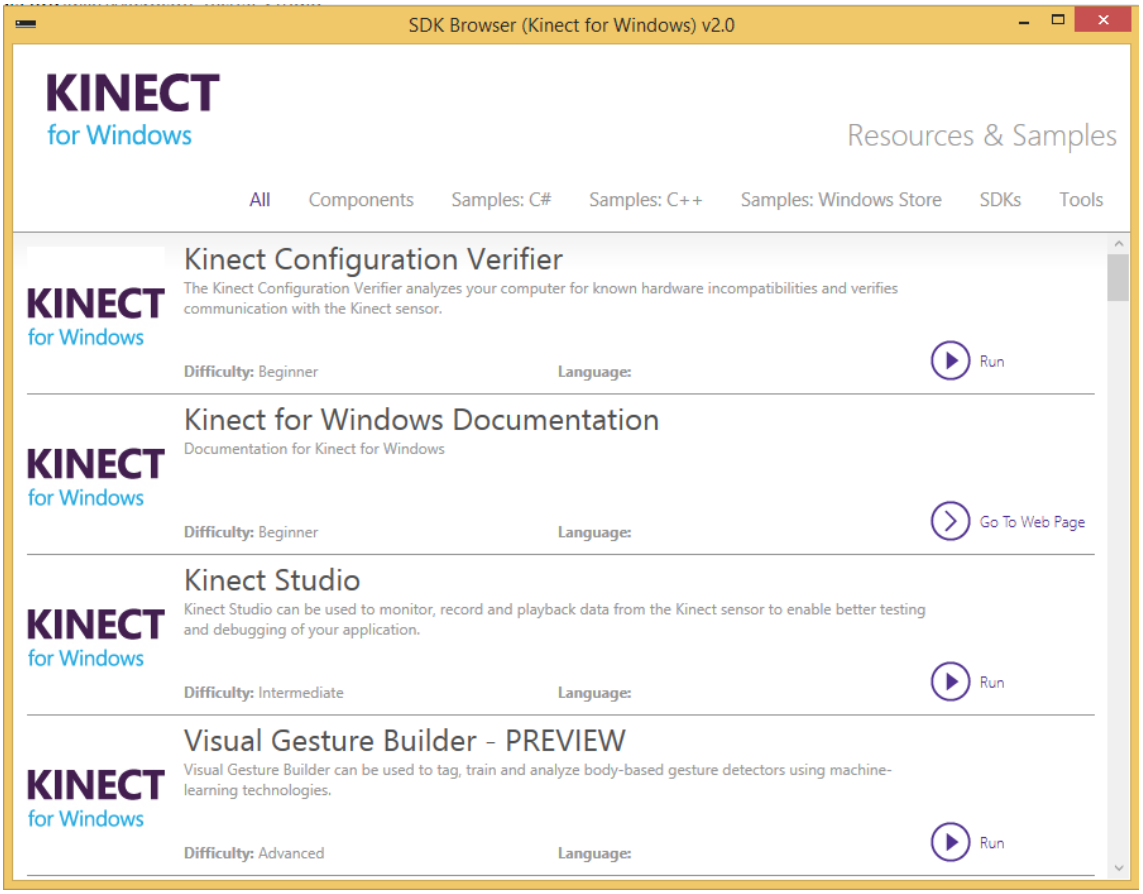

Fig. 5.5 *Programa SDK Browser*

Un conjunto de las aplicaciones más interesantes que nos proporciona el SDK es:

- **Audio Basics:** Nos proporciona información acerca del sonido del entorno y de la dirección de la que proviene ese sonido (ver figura 5.6).
- **Body Basics:** Detecta los cuerpos humanos y nos los muestra como una unión de 26 líneas (ver figura 5.7).
- **Body Index Basics:** Detecta los cuerpos humanos y nos muestra sus contornos (ver figura 5.8).
- **Color Basics:** Programa para captar y representar imágenes RGB a 15 o 30 fps.
- **Infrared Basics:** Programa para captar imágenes a través de los sensores infrarrojos y representarlas a 30 fps (ver figura 5.9).
- **Depth Basics:** Detecta la profundidad respecto a Kinect a la que se encuentran los objetos y nos la representa mediante imágenes a 30 fps (ver figura 5.10).
- **Coordinate Mapping Basics:** Detecta los cuerpos y realiza algo similar a un Chroma Key, sustituyendo el fondo por una imagen estática (ver figura 5.11).
- **Face Basics:** Detecta nuestra cara y nos da información de sus rotaciones, si lleva gafas, si está feliz etc. (ver figura 5.12).
- **HD Face Basics:** Genera un modelo 3D de la cabeza. Hablaremos más detalladamente de esta aplicación en capítulos posteriores.

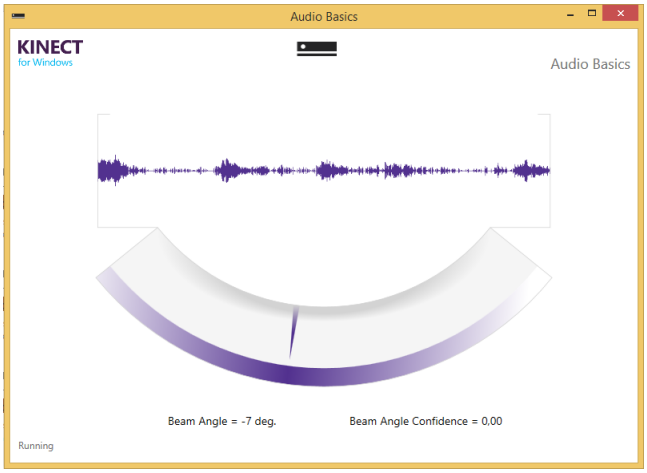

Fig. 5.6 *Audio Basics*

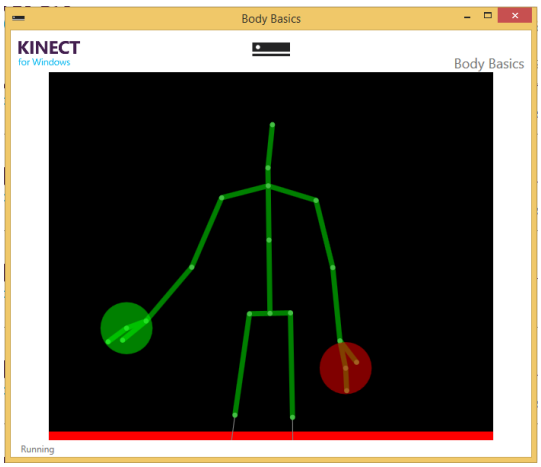

Fig. 5.7 *Body Basics*

Fig. 5.8 *Body Index Basics*

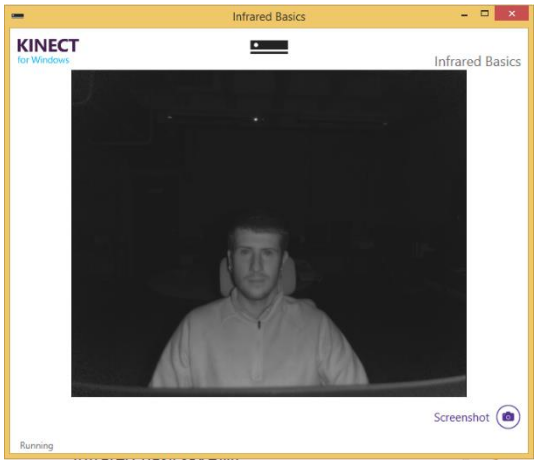

Fig. 5.9 *Infrared Basics*

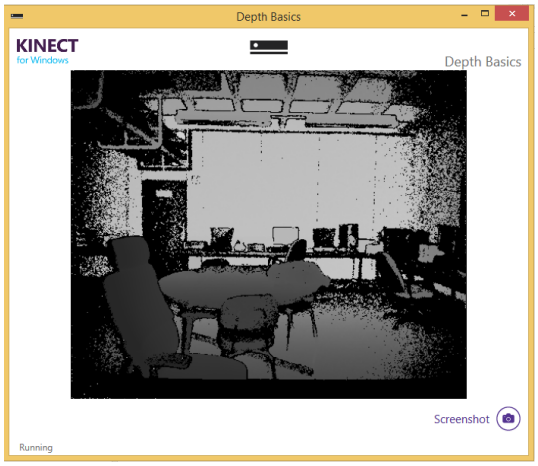

Fig. 5.10 *Depth Basics*

Fig. 5.11 *Coordinate Mapping Basics*

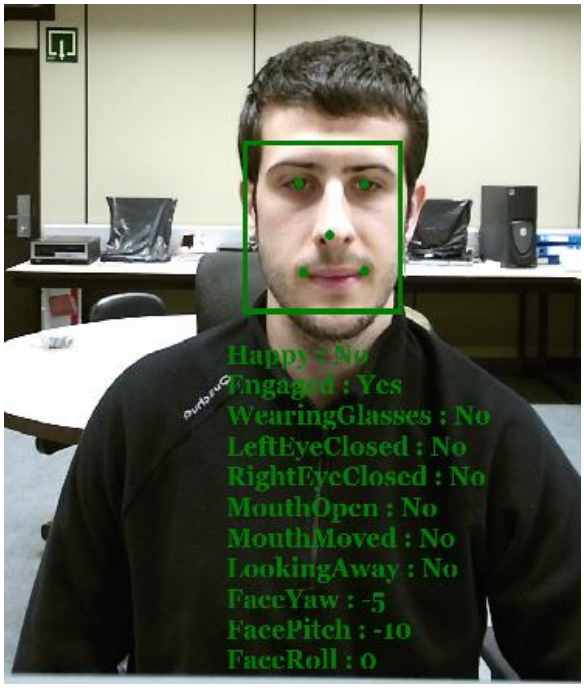

Fig. 5.12 *Face Basics*

5.5 Basel Face Model

El Basel Face Model (BFM) es un proyecto llevado a cabo por el Departamento de Ciencias de la Computación de la Universidad de Basel (Suiza). Un objetivo de este proyecto es el de facilitar el acceso a modelos deformables de calidad a un mayor grupo de investigadores; tal y como dicen en su página web: "*To allow a larger group of researchers access to high quality Morphable Models, we make the Basel Face Model available on this website*" **[6]**.

Para realizar un modelo medio deformable, la Universidad de Basel realizó un escaneo 3D de 100 caras de hombres y 100 caras de mujeres. La edad de las personas escaneadas estaba comprendida entre los 8 y los 62 años con una media de 24.97 años. Estos escaneos fueron reparametrizados para conseguir que cada posición en los vectores de datos corresponda al mismo punto en cualquier cara (ver figura 5.13) **[6].**

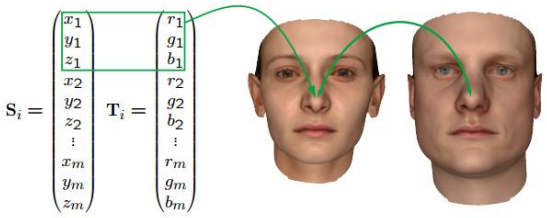

Fig. 5.13 *Tras la reparametrización, cada punto (en este caso la punta de la nariz) tiene la misma posición en el vector de datos para la totalidad de los escaneos* **[6]**.

Una vez realizada la reparametrización, se creó un modelo de cara media a partir de los 200 escaneos. Podemos ver el resultado de esta cara media en la figura 5.14.

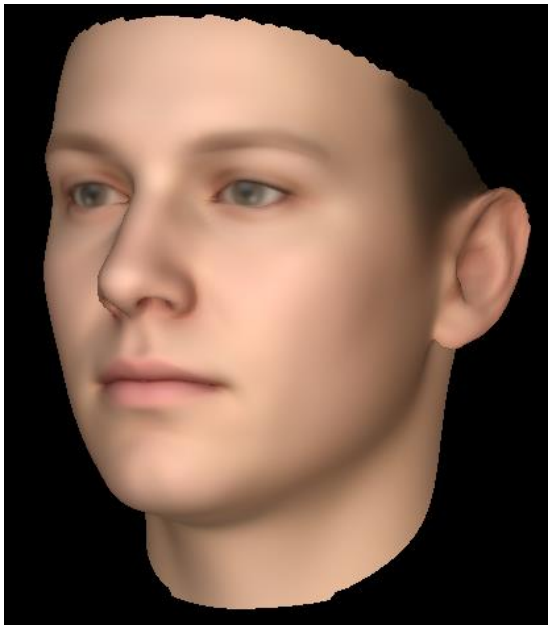

Fig. 5.14 *Modelo de la cara media obtenida a partir de las 200 caras escaneadas y reparametrizadas.*

Mediante la descarga del BFM disponible desde la página web del proyecto tenemos acceso, entre otras cosas, a los datos del BFM y al código de MATLAB que nos permitirá trabajar con estos datos.

En las figuras 5.15 y 5.16 vemos la nube de puntos del modelo de la cara media. Se trata de una matriz de 53490x3 puntos (53490 puntos con valores de “x”, “y” y “z”).

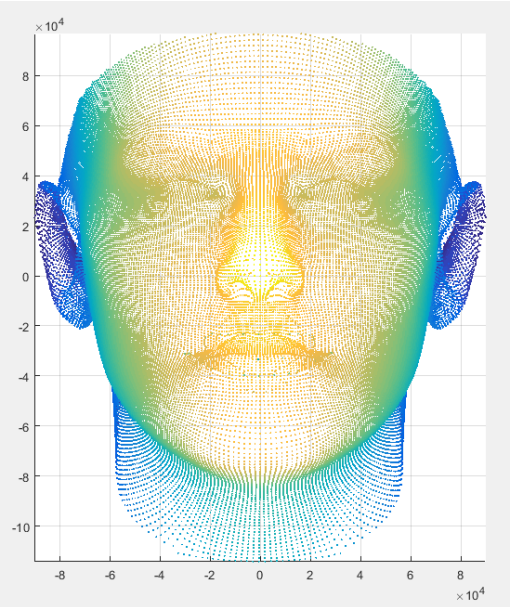

Fig. 5.15 *Nube de puntos del modelo deformable*

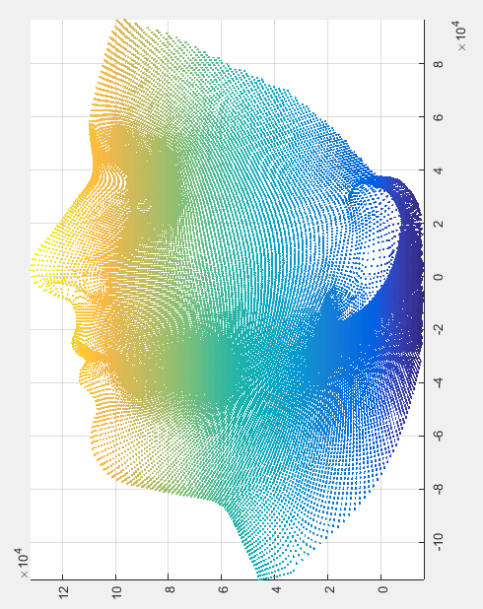

Fig. 5.16 *Perfil de la nube de puntos del modelo deformable*

Este código de MATLAB nos permite realizar modificaciones al modelo deformable de la cara media a través de la modificación de sus componentes principales. Los componentes principales del modelo deformable son las 199 variables a través de las cuales se pueden modificar las características del modelo. Cada una de ellas modifica al modelo de una forma diferente; algunas modifican el tamaño del modelo; otras la anchura; otras la separación de los ojos etc.

En la figura 5.17 vemos como modificando los tres primeros componentes principales podemos modificar el modelo deformable y generar nuevas caras.

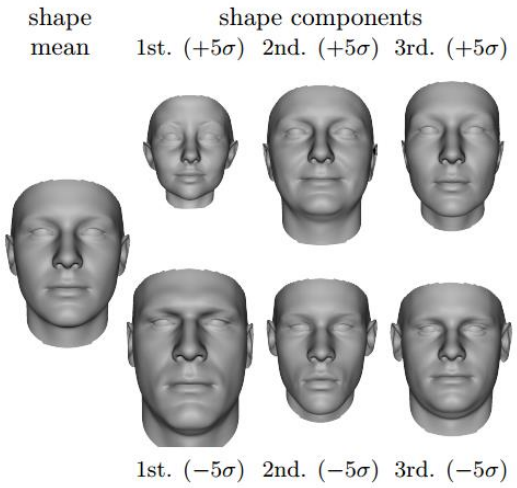

Fig.5.17 *Diferentes modificaciones del modelo deformable mediante el ajuste de los 3 primeros componentes principales* **[6]**.

También es posible modificar las texturas del modelo medio pero en este proyecto se ha decidido mantener las del modelo medio.

Utilizaremos los archivos del BFM para generar el modelado 3D de la cabeza obtenida por Kinect modificando los componentes principales del modelo deformable. Para ello, haremos uso de las múltiples funciones incluidas en estos archivos obtenidos a través de la página web del proyecto.

Capítulo 6
Procedimiento y Resultados

En este capítulo detallaremos los pasos que se han ido realizando a lo largo del proyecto; comenzando desde el tratamiento de la nube de puntos de Dasha, la programación del dispositivo Kinect, el tratamiento previo a la nube de puntos de Kinect y, por último, el ajuste del modelo.

Ya que vamos a trabajar con modelos 3D, es recomendable detallar el sistema de coordenadas que vamos a utilizar en nuestro proyecto. Las coordenadas "x" determinan la anchura de la cara, las "y" la altura de la misma y las "z" la profundidad. En la figura 6.1 se detalla gráficamente este sistema.

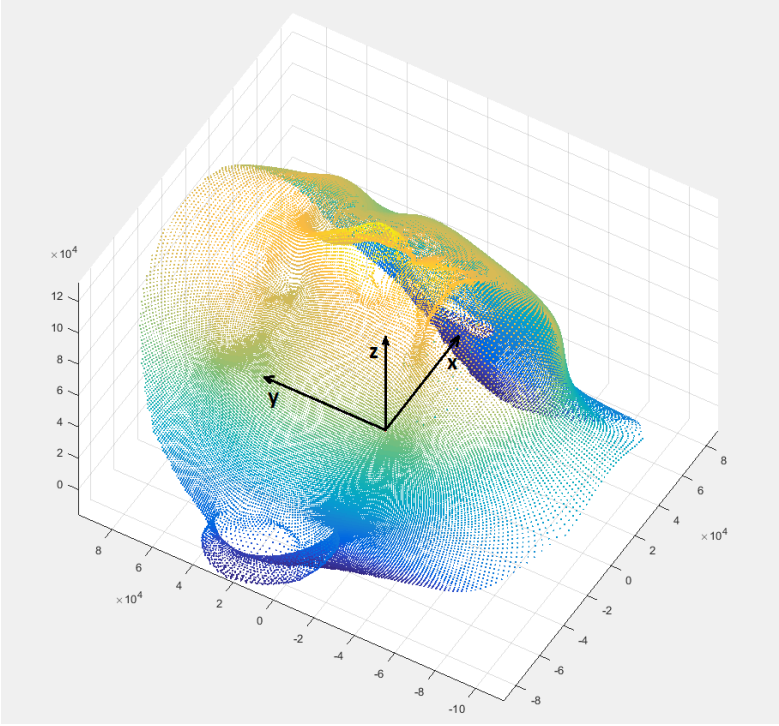

Fig. 6.1 *Sistema de coordenadas a utilizar*

Una vez aclarado este punto, pasemos a describir el procedimiento.

6.1 Dasha

Cuando este proyecto comenzó, el Laboratorio de Proyectos, Teoría de la señal y Comunicaciones, tenía un dispositivo Kinect a su disposición pero carecía del adaptador necesario para que Kinect se pudiese utilizar en el PC; por lo que mientras el adaptador llegaba, se decidió ir realizando pruebas con la nube de puntos de Dasha.

La nube de puntos de Dasha es una matriz de 100174x3 puntos obtenida, como ya se ha comentado en los capítulos 3 y 4, mediante un escaneo láser; por lo que la cantidad de puntos, el sistema de referencia, la escala etc. tendrán que ser tratados para ajustarlos a los del modelo BFM. En las figuras 6.2 y 6.3 podemos ver la nube de puntos resultante del escaneo 3D de la cabeza Dasha.

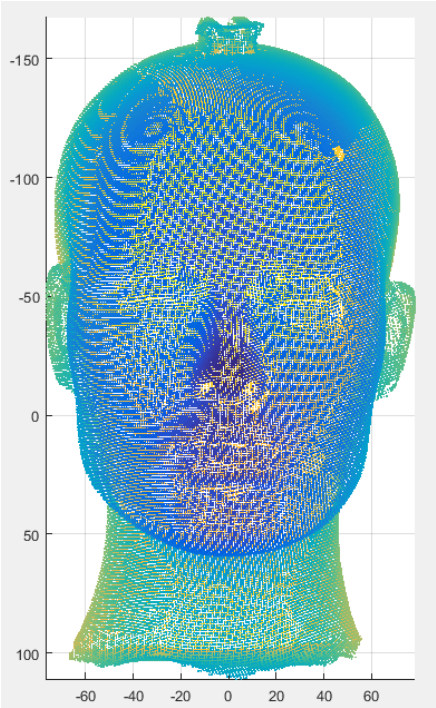

Fig. 6.2 *Nube de puntos Dasha*

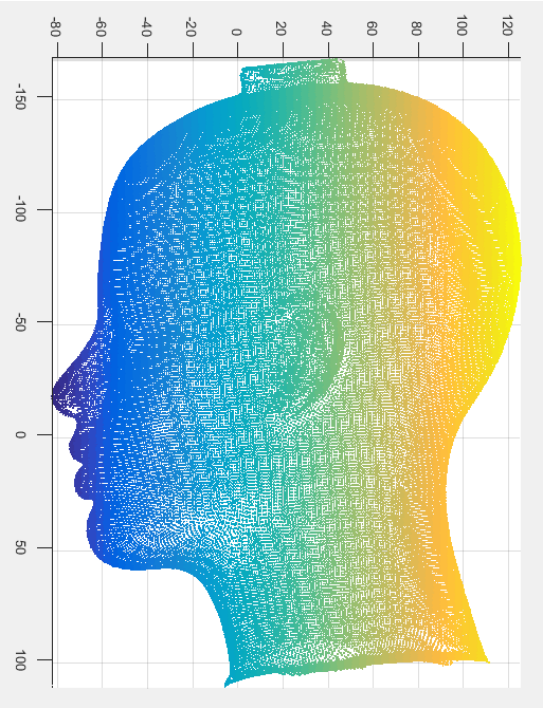

Fig. 6.3 *Perfil de la nube de puntos de Dasha*

Se ha tomado la decisión de incluir el trabajo realizado con la nube de puntos de Dasha ya que se considera que fue una parte muy importante en el proceso de toma de contacto con el proyecto y sirvió para familiarizarse con los conceptos relacionados con los modelos 3D y con los algoritmos básicos de tratamiento de nubes de puntos.

6.1.1 Tratamiento de la nube de puntos de Dasha

Como se ha visto en las figuras 6.2 y 6.3 la nube de puntos de Dasha tiene valores para todo el contorno de la cabeza y no solo para la zona de la cara. Esto supone un problema ya que los archivos del BFM trabajan únicamente con la zona facial; si queremos mantener una relación entre ambas nubes de puntos, necesitamos eliminar los puntos de Dasha que no mantengan relación con los del BFM.

Para ello utilizamos una función que recortase las zonas de puntos no deseadas de Dasha y dejase únicamente la zona facial. Esto lo conseguimos determinando los valores de las coordenadas “x”, “y” y “z” a partir de los cuales no necesitábamos más puntos. El resultado se puede ver en las figuras 6.4 y 6.5.

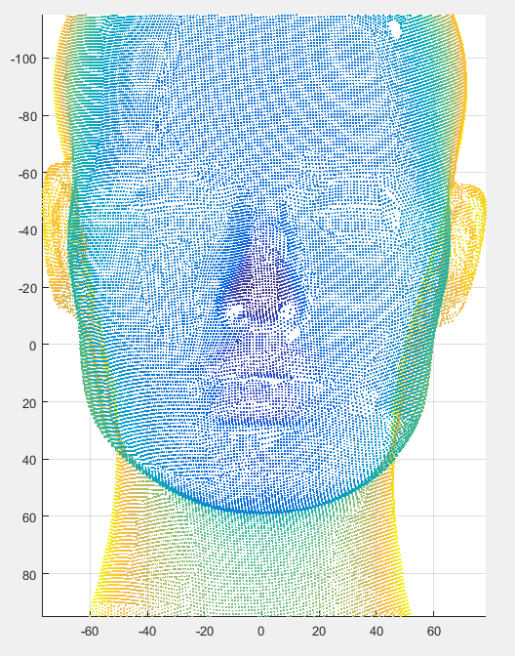

Fig. 6.4 *Nube de puntos de Dasha tras ser recortada*

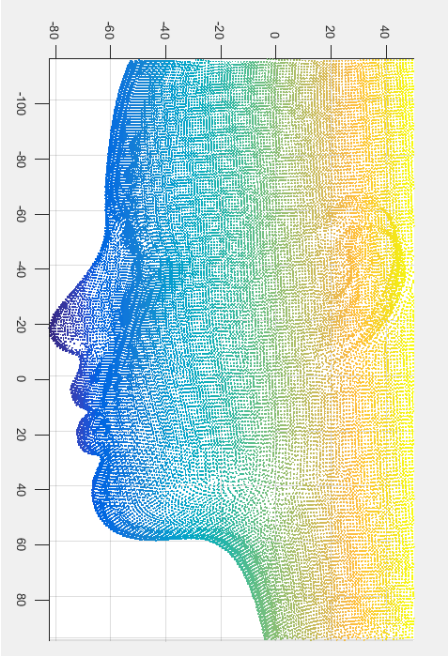

Fig. 6.5 *Perfil de la nube de puntos de Dasha tras ser recortada*

Otro problema es el hecho de que la nube de puntos de Dasha no comparte el sistema de coordenadas que hemos descrito en la introducción del capítulo sino que sus ejes “y” y “z” están invertidos (ver figura 6.6). Por lo tanto, necesitamos implementar otra función que transforme el sistema de coordenadas de Dasha para ajustarlo al del BFM.

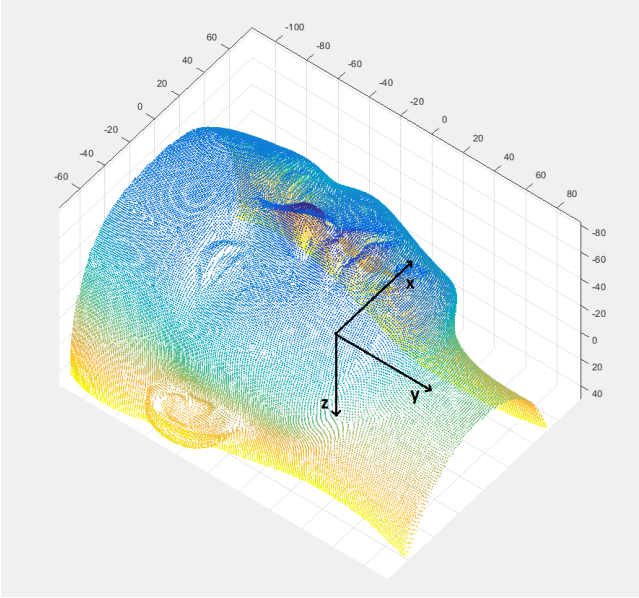

Fig. 6.6 *Sistema de coordenadas de Dasha*

Para ello, utilizaremos la matriz de Rotación-Traslación de la ecuación (1) en la cual el valor de traslación es nulo.

$$RT=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(1)}$$

Aplicándosela a la nube de puntos de Dasha, conseguimos corregir los ejes invertidos tal y como vemos en (2).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} * \begin{bmatrix} x \\ -y \\ -z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \textbf{(2)}$$

Una vez hecho esto, solo nos quedan dos últimos pasos para terminar de tratar a la nube de puntos de Dasha y ajustarla al modelo BFM. El primero de ellos es el de reescalarla; la nube de puntos de Dasha es mil veces más pequeña que el modelo BFM por lo que tuvimos que aplicar un escalado de 1000:1. Por último, existe una incoherencia entre los orígenes de ambos modelos por lo que es necesario unificar el origen de coordenadas de ambos modelos. En este punto surgieron muchas dificultades y al final, y aun sabiendo las consecuencias negativas que esto acarrearía en los resultados, se decidió no implementarlo.

En la figura 6.7 podemos observar el resultado final tras aplicar todos estos ajustes.

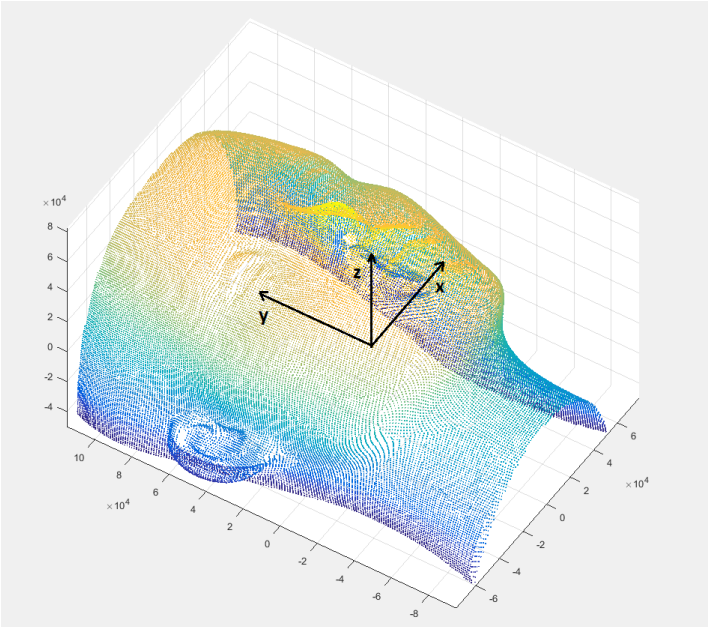

Fig. 6.7 *Nube de puntos de Dasha tras su tratamiento para ajustarla al modelo BFM*

6.1.2 Ajuste del modelo BFM con la nube de puntos de Dasha

Una vez que ya tenemos la nube de puntos de Dasha adaptada a las características del modelo BFM, el siguiente paso es realizar el ajuste del modelo deformable; para ello, y como ya hemos visto en el capítulo 5, tenemos que modificar las componentes principales del modelo deformable hasta adaptarlo al modelo de Dasha. Esto se consigue mediante el uso de un vector de tamaño 199x1 en el que cada una de las celdas modifica un componente principal. Cabe destacar que no hace falta modificar todos los componentes principales para obtener una nueva cara sino que se puede modificar la cantidad de ellos que se desee. En nuestro caso utilizamos un bucle de 12 iteraciones en el que cada una de ellas utiliza una cantidad distinta de componentes principales; empezando por los 5 primeros y continuando por 10, 15, 20, 30, 40, 50, 60, 80, 100, 150 hasta acabar en los 199.

Una vez tenemos este bucle, se utiliza una función que, a su vez, realiza nuevas iteraciones en las que se van modificando los componentes principales que se están utilizando para, mediante distintos procesos matemáticos, obtener la combinación de ellos que mejor adapte el modelo BFM al nuestro. Por otro lado, hay que tener en cuenta que los últimos componentes principales generan componentes de alta frecuencia en nuestros modelados por lo que no está de más contemplar la posibilidad de regularizar ese proceso matemático para suavizar el efecto producido por estos componentes principales.

Inicialmente se optó por no regularizar el proceso matemático y surgió la idea de utilizar un algoritmo de ICP (Iterative Closest Point) para obtener la mejor combinación de componentes principales. El ICP es un algoritmo cuyo propósito es minimizar la diferencia entre dos nubes de puntos. El algoritmo en cada paso de iteración, primero calcula las correspondencias de puntos usando el criterio del punto más cercano, y después calcula la transformación rígida que minimiza la función de coste para dichas correspondencias. La función de coste a minimizar es la siguiente (3):

$$E(R,T) = \sum_{i=1}^{Nm}\sum_{j=1}^{Nd} \omega_{i,j} \left\| m_i - (Rd_j + T) \right\| \qquad \textbf{(3)[7]}$$

Donde $\boldsymbol{Nm}$ y $\boldsymbol{Nd}$ representan la cantidad de puntos de cada una de las dos nubes de puntos, $\boldsymbol{m_i}$ y $\boldsymbol{d_j}$ los puntos en forma de vector (x,y,z) y $\boldsymbol{\omega_{i,j}}$ las correspondencias de puntos; 1 si $\boldsymbol{m_i}$ empareja con $\boldsymbol{d_j}$; 0 en otro caso. Cabe destacar que en nuestro caso, aproximamos la norma como una suma de los valores absolutos de las diferencias de cada componente vectorial.

Los resultados obtenidos en un primer experimento no fueron muy satisfactorios debido a que, por un problema que en ese momento desconocíamos y que más adelante detallaremos, el proceso no se llegó a completar, realizando únicamente los cálculos para el uso de 5, 10, 15, 20, 30, 40, 50 y 60 componentes principales. En la figura 6.8 podemos observar el error que existe entre los puntos de las cabezas generadas por nuestro algoritmo y los puntos de la cabeza original para cada número de componentes principales (o modos) utilizados.

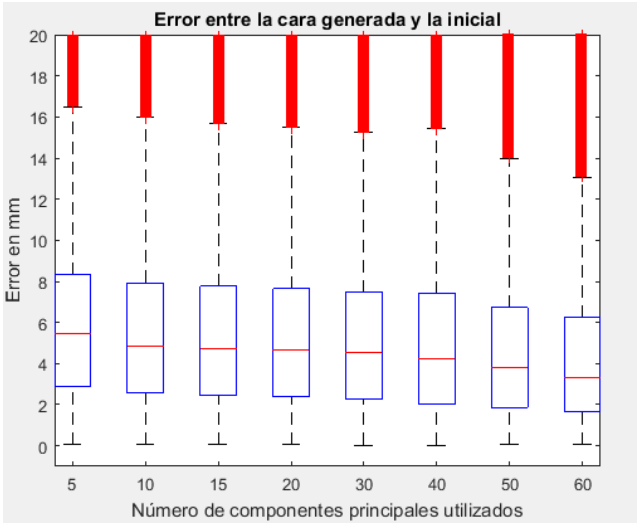

Fig. 6.8 *Error, en mm, entre la cara generada y la original para cada número de componentes principales utilizados en el primer experimento*

Se puede observar que, en los primeros "pasos", el error se reduce ligeramente pero al aumentar el número de modos a un valor superior a 40, el error se reduce más significativamente. Se han acotado los resultados a valores de error de entre -1 y 20 mm debido a la existencia de outliers con valores de error de hasta 60 mm, lo que dificultaba el análisis de la zona que nos interesa.

Podemos también sacar conclusiones a través de la visualización directa de las caras generadas; en la figura 6.9 se muestran las caras generadas por este primer experimento.

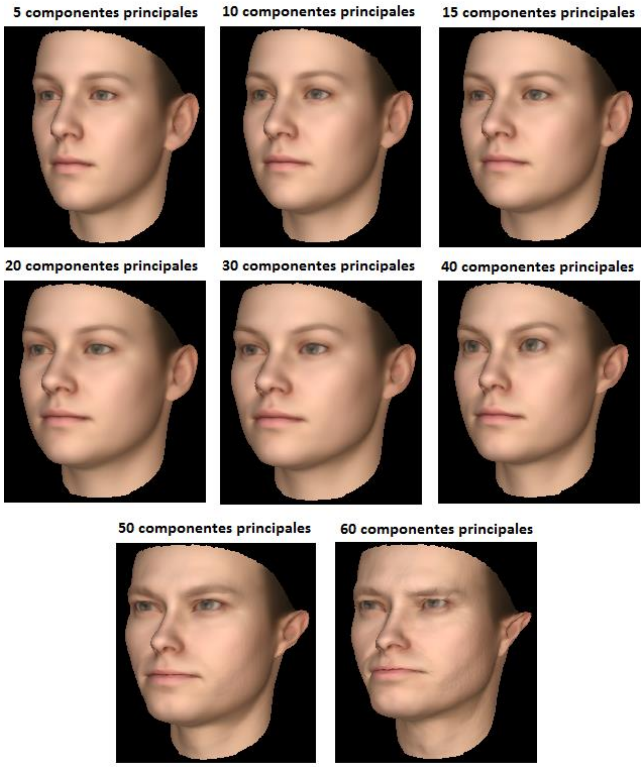

Fig. 6.9 *Cabezas generadas por el primer experimento*

Se observa que existe muy poca diferencia entre los modelos generados en las 5 primeras iteraciones y el modelo medio (ver figura 5.14). También podemos comprobar como al aumentar el número de componentes principales, se comienzan a percibir las altas componentes frecuenciales en los modelos.

En este punto y para agilizar los tiempos de simulación se decidió realizar un diezmado aleatorio tanto a la nube de puntos de Dasha como al modelo BFM en el que nos quedamos únicamente con el 10% de los puntos totales; como era de esperar, esto agilizó el proceso. Esta vez no ocurrió ningún error, completándose así el proceso para las 12 iteraciones del bucle. Del mismo modo que en el experimento anterior, en la figura 6.10 podemos ver el error que existe entre los puntos de las cabezas generadas por nuestro algoritmo y los puntos de la cabeza original para cada número de componentes principales utilizados.

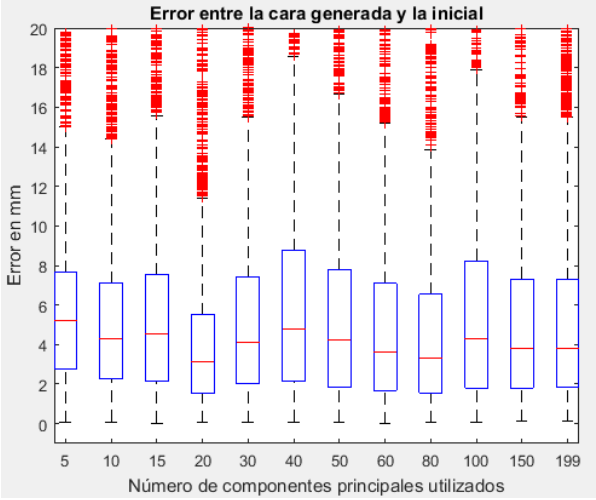

Fig. 6.10 *Error, en mm, entre la cara generada y la original para cada número de componentes principales utilizados en el segundo experimento*

En esta ocasión, los resultados no siguen un patrón muy claro por lo que el análisis de los mismos no es algo sencillo. Sin embargo, los resultados visuales (ver figura 6.11) nos proporcionaron información clave para el desarrollo del proyecto:

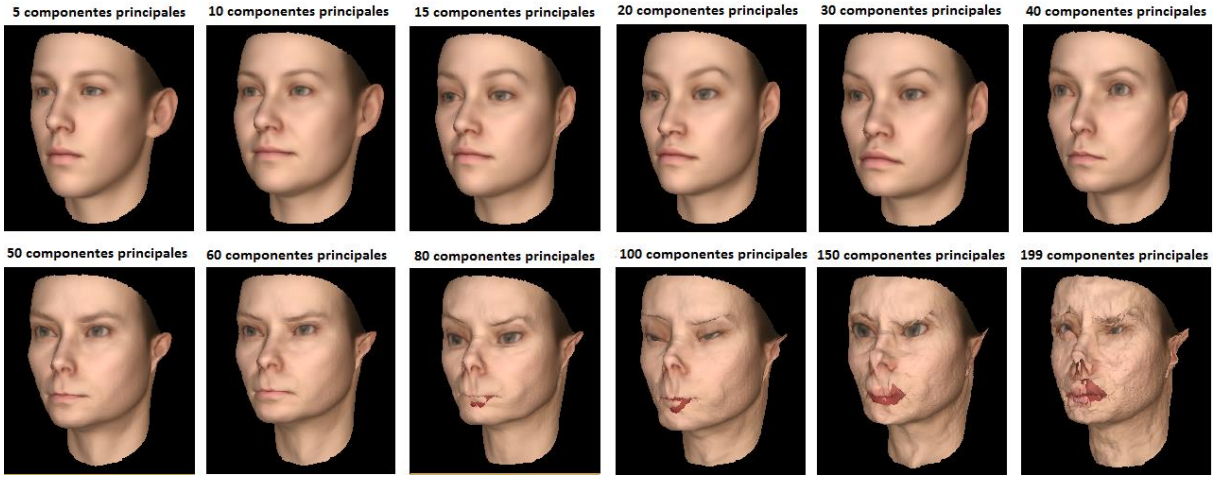

Fig. 6.11 *Cabezas generadas por el segundo experimento*

La aparición de altas componentes frecuenciales en las últimas iteraciones de esta simulación es perfectamente visible y afecta de manera muy significativa al resultado final; por lo que fue en este punto donde se determinó totalmente necesario el uso de la regularización. Para ello, generamos un nuevo bucle de 11 iteraciones en el que cada iteración utiliza un valor de regularización distinto. Estos valores de regularización son: 0, 100, 200, 500, 1000, 2000, 3500, 5000, 7500, 10000 y 20000. Hay que tener en cuenta que en los siguientes experimentos, obtendremos 11 resultados (uno por cada valor de regularización) para cada número de componentes principales utilizados, es decir, 12*11= 132 resultados. Esto hará que la evaluación de los resultados sea algo más compleja.

Una vez que se decidió implementar la regularización se realizó un nuevo experimento en el que, de nuevo, se decidió realizar un diezmado aleatorio tanto a la nube de puntos de Dasha como al modelo BFM; esta vez nos quedamos con el 10%. El experimento no se pudo completar ya que se repitió el mismo problema que tuvimos en el primer experimento. Tras analizar los scripts y las funciones utilizadas, se dio con el problema; el problema consistía en que, dentro del algoritmo del ICP existe una función que, mediante el uso de un épsilon, determina si la transformación realizada es rigida o no y en el caso de que detecte que no es rígida, detenga el proceso. Este error se corrigió aumentando en un factor 10 el valor de esa épsilon y se realizó un nuevo experimento utilizando una nueva función de diezmado llamada "*pcdownsample*" que, en principio, se creía que iba a proporcionar mejores resultados. Mediante esta función, nos quedamos con el 10% de los puntos. Los resultados fueron no fueron aceptables, ya que los modelos generados fueron prácticamente la cara media. Se pensó que la razón podría ser un diezmado excesivo así que se realizó otro experimento; esta vez quedándonos con el 50% de los puntos. Los resultados fueron los mismos así que se decidió utilizar otra técnica de diezmado.

El algoritmo de diezmado que se usó a partir de ese momento fue "*MyCrustOpen*". Este algoritmo no necesitaba como valor de entrada una nube de puntos sino que requería de una superficie triangulada. Así pues, tuvimos que crear una superficie triangulada a partir de la nube de puntos de Dasha; tras varios intentos dimos con una solución razonablemente buena obteniendo el resultado que se muestra en la figura 6.12.

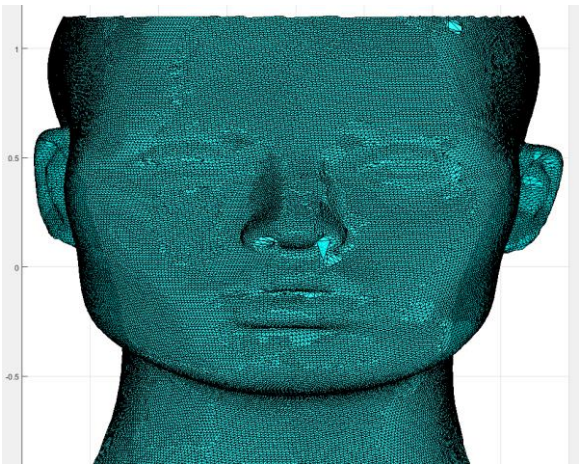

Fig. 6.12 *Triangulación de Dasha*

A partir de esta superficie triangulada, realizamos el nuevo diezmado quedándonos únicamente con el 10% de los puntos de Dasha. Realizamos un nuevo experimento con este valor de diezmado y obtuvimos los errores medios mostrados en las figuras 6.13 y 6.14. A partir de este momento se ha decidido cambiar la forma de evaluar y mostrar los resultados ya que tenemos 11 (uno por cada valor de regularización) para cada número de modos utilizados.

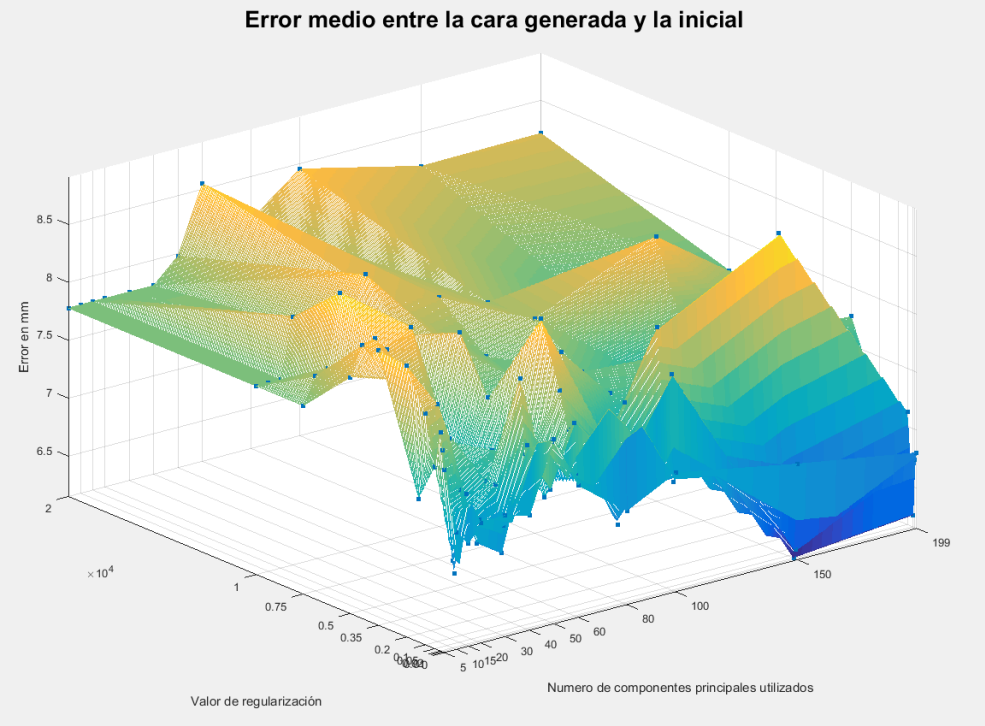

Fig. 6.13 *Error medio obtenido para los diferentes valores de regularización y de componentes principales utilizados*

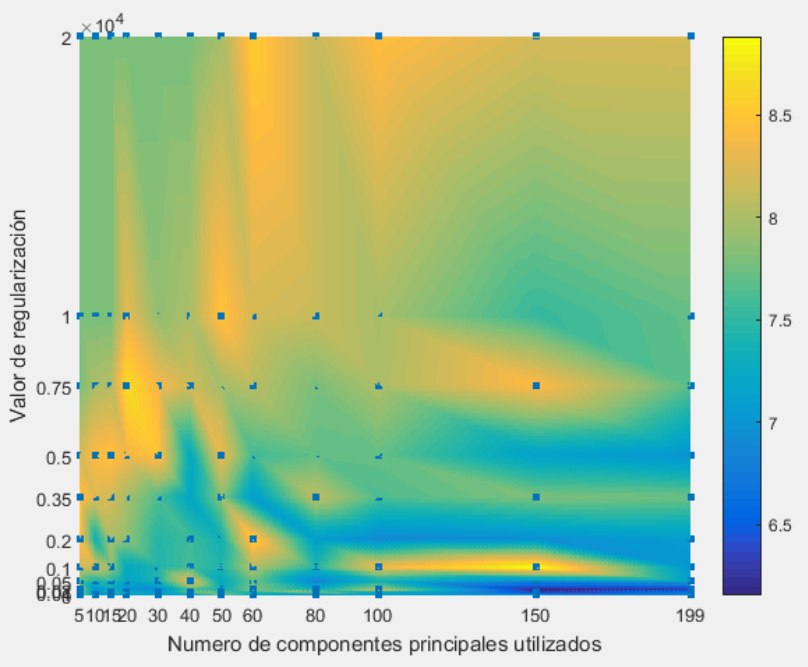

Fig. 6.14 *Error medio, visto desde arriba, obtenido para los diferentes valores de regularización y de componentes principales utilizados*

Como se puede observar, los valores de errores medios más pequeños se encuentran en las zonas con valores de regularización bajos y, sobretodo, con el uso de un gran número de componentes principales. Esto tiene sentido ya que cuantas más componentes principales, más podremos deformar el modelo medio y adaptarlo al nuestro. Sin embargo, existen ciertas anomalías entre estos resultados y los obtenidos gráficamente. En la figura 6.15 podemos ver cómo, para valores de regularización bajos y un uso de 199 componentes principales, las caras generadas presentan altas frecuencias y poco tienen que ver con la cara original.

En esta figura también se puede observar que, mediante el uso de valores de regularización elevados, se consiguen eliminar las componentes frecuenciales altas. Hay que fijarse que si utilizamos valores de regularización muy elevados obtenemos caras medias.

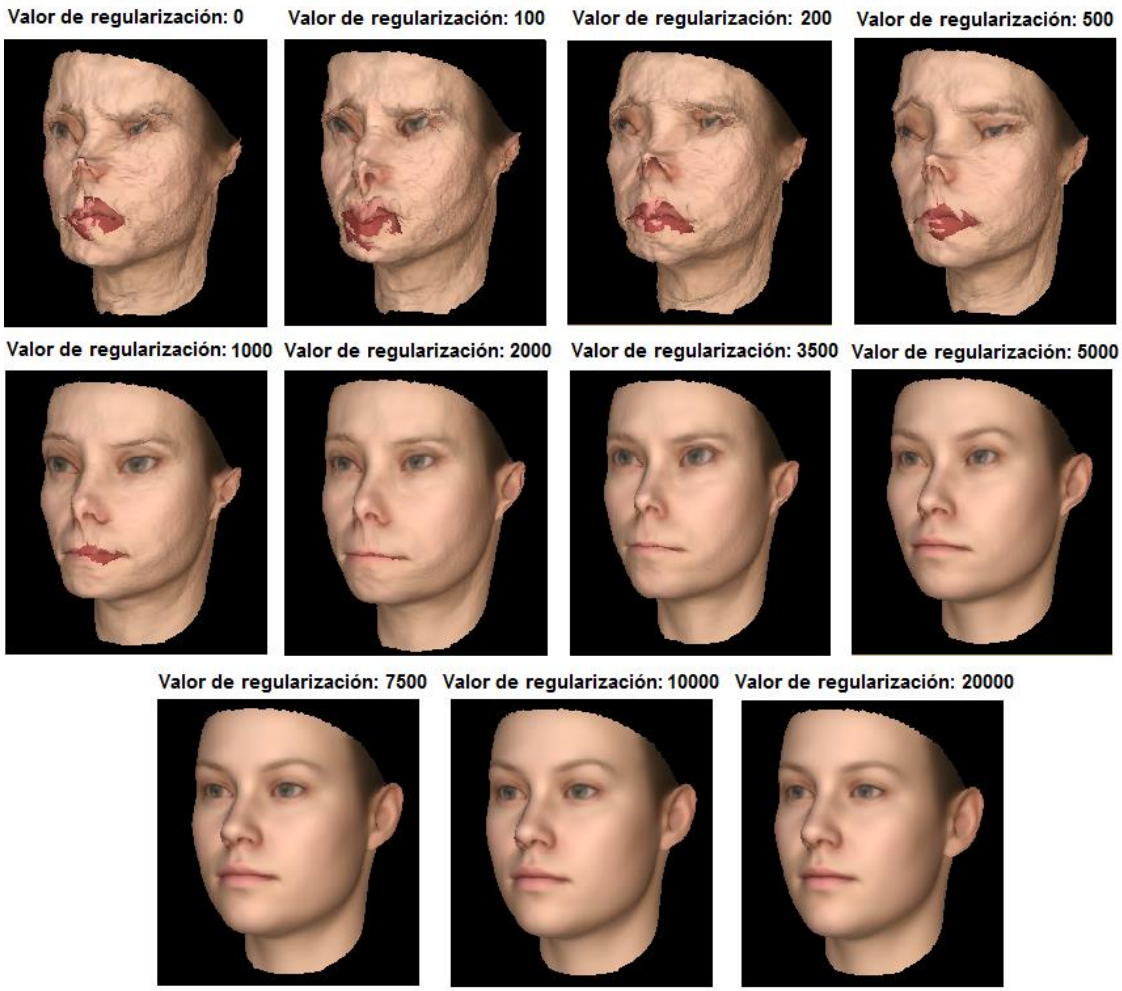

Fig. 6.15 *Caras generadas a partir del uso de 199 componentes principales y de los diferentes valores de regularización*

Fue en este punto; varias semanas después de comenzar el proyecto, cuando el Laboratorio de Proyectos, Teoría de la señal y Comunicaciones recibió el adaptador de Kinect para PC por lo que ya pudimos comenzar a trabajar en el verdadero proyecto.

6.2 Programación Kinect

Lo primero que se hizo fue instalar el SDK de Kinect y, mediante una aplicación descargada desde el SDK, comprobar que el ordenador cumplía los requisitos necesarios para el correcto funcionamiento de Kinect. Una vez comprobado que todo estaba correcto, se procedió a descargar algunos de los programas ejemplo incluidos en el SDK. En el SDK de Kinect podemos encontrar los programas en 2 versiones; una escrita en C++ y otra escrita en C#. Como ya se ha comentado en el capítulo 5 nosotros escogimos el lenguaje C#. A partir del código de estos programas se intentaron comprender las bases de la programación de Kinect para, más adelante, poder realizar las modificaciones pertinentes en ellos y obtener un programa que realizase la labor que nos conviene para este proyecto.

El programa ejemplo que se utilizó como base para crear la aplicación que nos proporcionase la nube de datos es el HDFaceBasics. A partir de imágenes de nuestra cabeza como son los perfiles (ver figuras 6.16 y 6.17) el frente (ver figura 6.18) y una leve inclinación hacia arriba (ver figura 6.19) HDFaceBasics genera un modelado 3D de nuestra cabeza (ver figura 6.20). En esta última figura se puede ver como el modelo de la cabeza es diferente respecto a las anteriores figuras.

La parte del programa que realiza el modelado permanece intacta, pero se ha creado una función dentro del programa que detecta el momento en el que el modelo se ha creado y obtiene los valores de yaw pitch y roll (ver figura 2.1) que nos indican la orientación de la cabeza. Una vez hecho esto crea una nueva carpeta (en el directorio que nosotros hayamos especificado) en la que genera 3 nuevos archivos; uno con los datos de orientación de la cabeza; otro con una lista de los trios de índices que permiten realizar una triangulación del modelo y un último con la nube de puntos del modelo generado.

La modificación final del programa se guardó con el nombre: “**GetPointCloud**”.

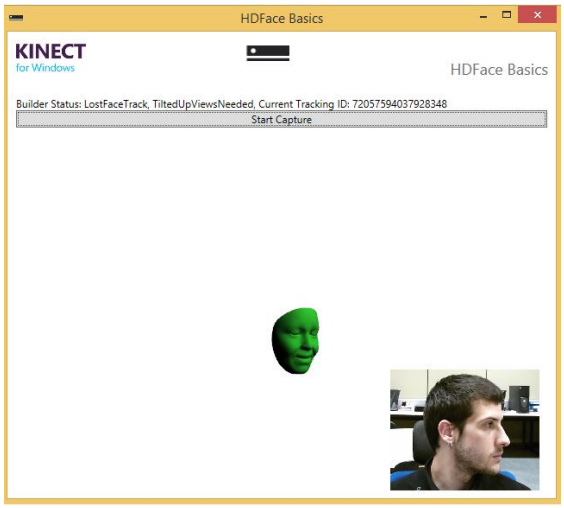

Fig 6.16 *Perfil derecho tomado por HDFaceBasics*

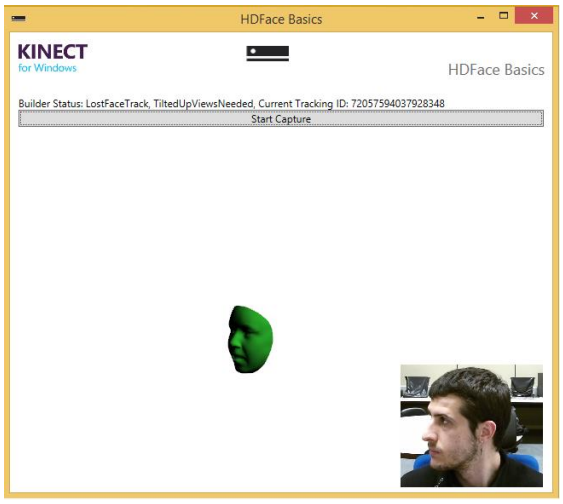

Fig 6.17 *Perfil izquierdo tomado por HDFaceBasics*

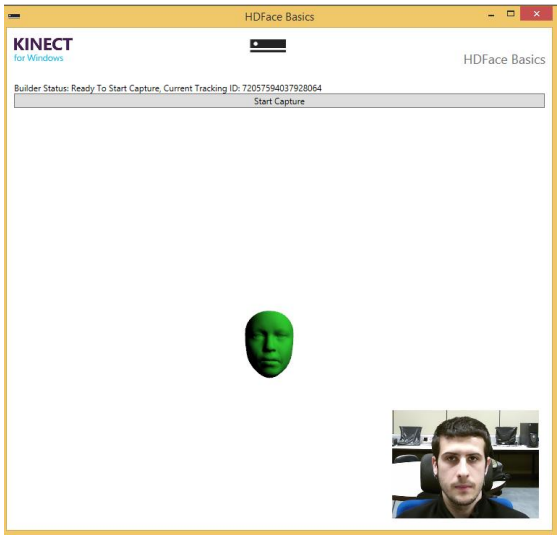

Fig 6.18 *Imagen frontal tomado por HDFaceBasics*

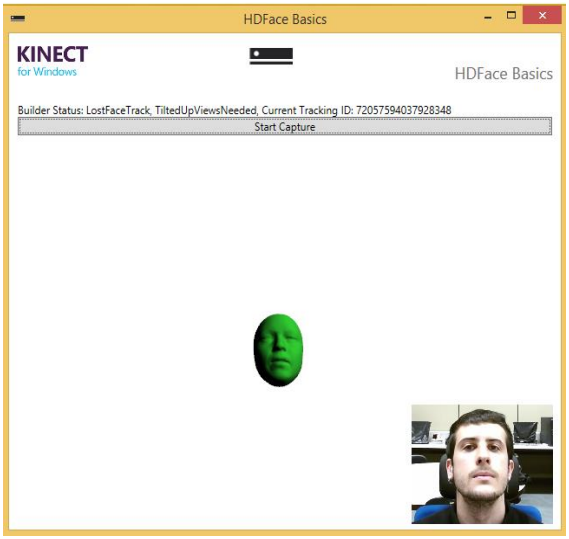

Fig 6.19 *Imagen frontal inclinada tomada por HDFaceBasics*

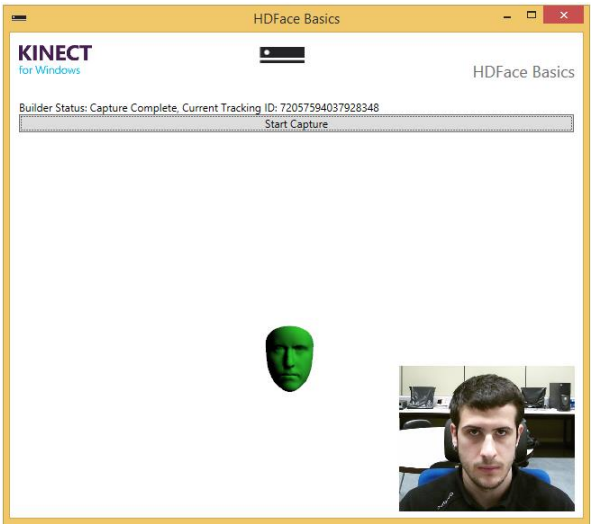

Fig 6.20 *Modelo creado por HDFaceBasics*

6.3 Tratamiento previo a la nube de puntos de Kinect

Una vez obtenida la nube de puntos mediante el programa modificado del apartado anterior, hubo que realizar un proceso parecido al comentado en el apartado 6.1.1.

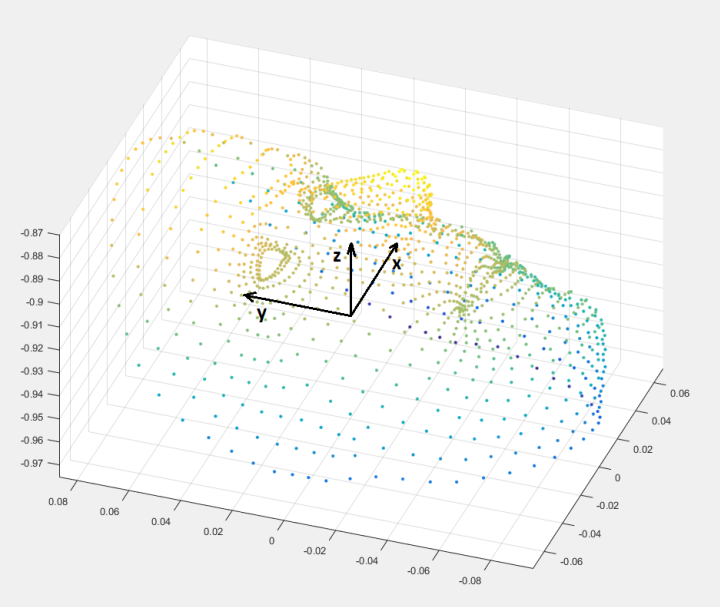

Fig. 6.21 *Nube de putos proporcionada por Kinect*

Como se puede ver en la figura 6.21 la nube de puntos proporcionada por Kinect comparte el mismo sistema de coordenadas que el modelo BFM. También se puede observar que es muchísimo menos densa que la de Dasha; en este caso la nube de puntos es una matriz de 1347x3. Sin embargo, tiene problemas de escala y orientación que deberán ser corregidos para ajustarlos al modelo BFM.

La figura 6.22 nos muestra el problema de orientación que acabamos de comentar. Como se ve, la cara no está perfectamente alineada con los ejes.

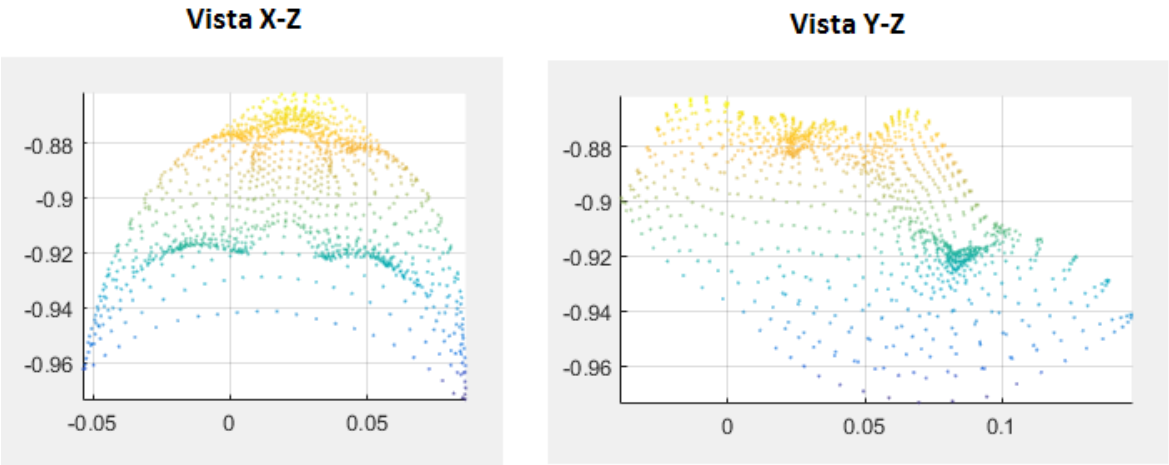

Fig. 6.22 *Problema de orientación de la nube de puntos de Kinect*

Inicialmente se trató de corregir este problema a través del uso del algoritmo de ICP pero, debido a que la diferencia de orientación entre la nube de puntos del BFM y la de Kinect era demasiado alta, este algoritmo no se comportaba correctamente. También se pensó en, a través de la programación de Kinect, obtener los valores de yaw, pitch y roll que nos indicasen la orientación de la cara en el momento de generar la nube de puntos. Se modificó el programa GetPointCloud para que lo hiciese pero, mientras se implementaba la solución en MATLAB, encontramos un documento en la API de Kinect y otro en la documentación del BFM que redirigieron esta parte del proyecto.

Estos documentos tratan sobre los puntos "clave" de cada una de las nubes de puntos, es decir, de los puntos que indican posiciones importantes de la cara como lo son la punta de la nariz, las esquinas de los ojos, las esquinas de los labios, etc. El documento de la API de Kinect nos proporciona una lista con los puntos clave de su nube de puntos; en la figura 6.23 podemos ver los puntos a los que se refiere, marcados en rojo.

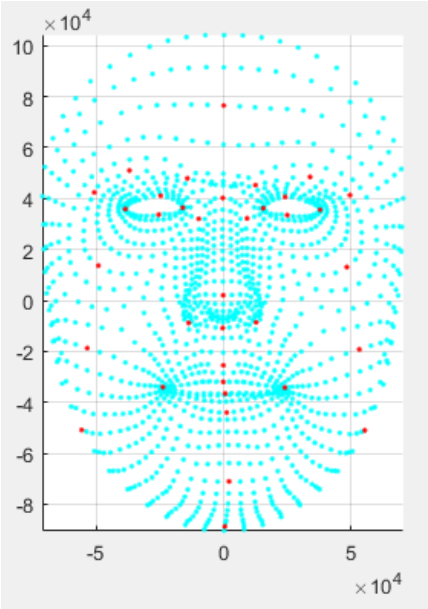

Fig. 6.23 *Keypoints de Kinect*

Por otro lado, en el archivo de la documentación del BFM se referencian los índices del modelo BFM a algunos de los puntos característicos del estándar MPEG-4 (ver figura 6.24).

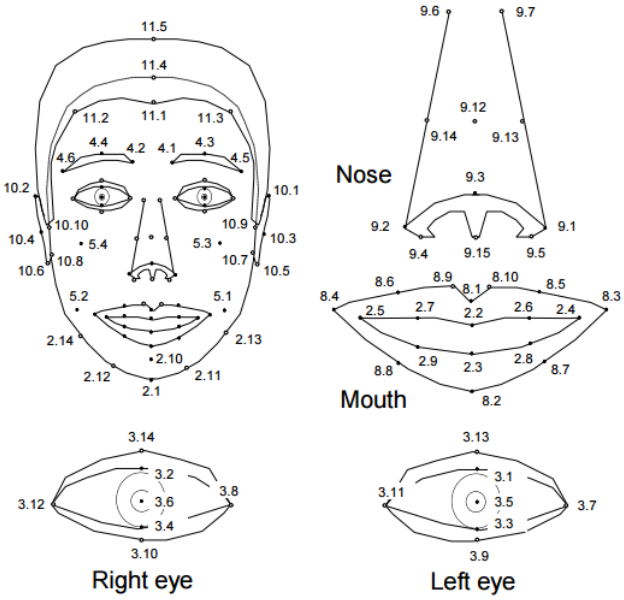

Fig. 6.24 *Puntos característicos del estándar MPEG-4* **[8]**

Del mismo modo que con la nube de puntos de Kinect; podemos ver, en rojo, los puntos contemplados en el modelo BFM (ver figura 6.25).

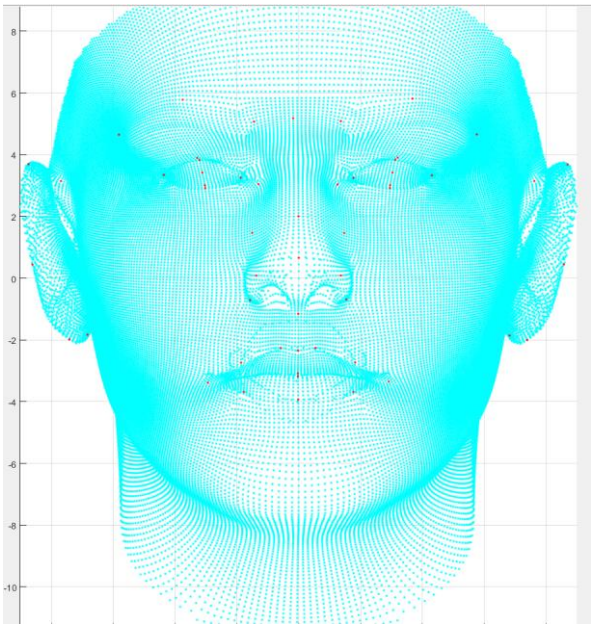

Fig. 6.25 *Puntos característicos del estandar MPEG-4 contemplados en el modelo BFM.*

A partir de estos datos pudimos obtener los pares de puntos coincidentes entre la nube de puntos de Kinect y la del BFM. Obtuvimos un total de 24 puntos; los cuales se pueden ver, tanto para el modelo BFM como para la nube de Kinect en la figura 6.26.

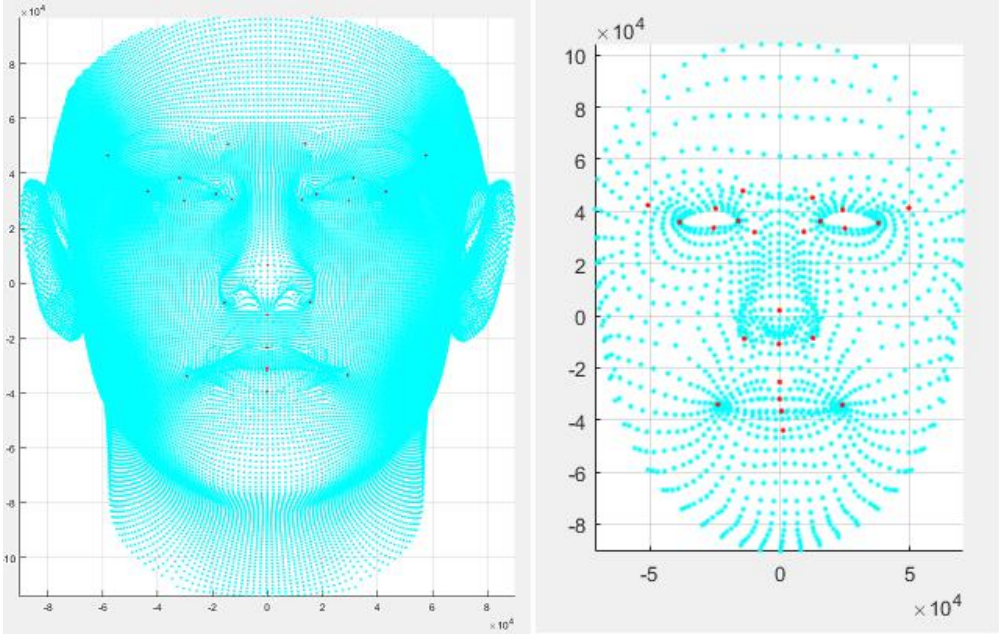

Fig. 6.26 *Listado de los 24 puntos coincidentes entre las nubes de puntos de Kinect y del BFM*

Una vez emparejados los puntos de ambas nubes, pudimos realizar un Procrustes de ellas y obtener así un alineamiento. En la figura 6.27 podemos ver un antes (izquierda) y un después (derecha)

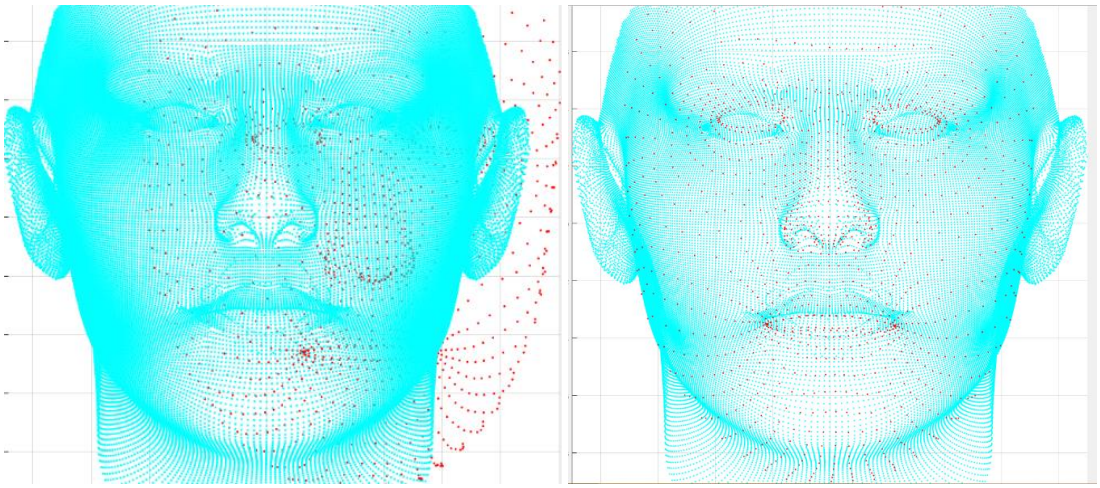

Fig. 6.27 *Antes (izquierda) y después (derecha) de realizar el Procrustes para orientar las nubes de puntos*

Al igual que en el proceso realizado a la nube de puntos de Dasha, existe una incoherencia entre los orígenes de los modelos BFM y Kinect; en este caso sin embargo, al haber realizado un Procrustes, las nubes de puntos se encuentran en una misma región espacial, es decir, la función ha realizado variaciones en los orígenes de las nubes de puntos. A pesar de ello, existe cierto error por lo que sería muy interesante el encontrar la forma de unificar los orígenes de ambos modelos.

Hecho esto, tenemos nuestra nube de puntos perfectamente adaptada para proceder a realizar el ajuste.

6.4 Ajuste del modelo

Al igual que se hizo con Dasha, una vez tuvimos la nube de puntos de Kinect ajustada al modelo BFM, el siguiente paso fue realizar, mediante los componentes principales del modelo BFM, una deformación del modelo medio. Esta vez se implementó regularización desde el principio.

En el primer experimento se optó por seguir utilizando el algoritmo de ICP para obtener la mejor combinación de componentes principales. No se realizó ningún diezmado ya que la nube de puntos de Kinect es menos densa que la de Dasha.; aun así, la simulación fue del orden de una semana. Los resultados obtenidos se muestran en la figura 6.28.

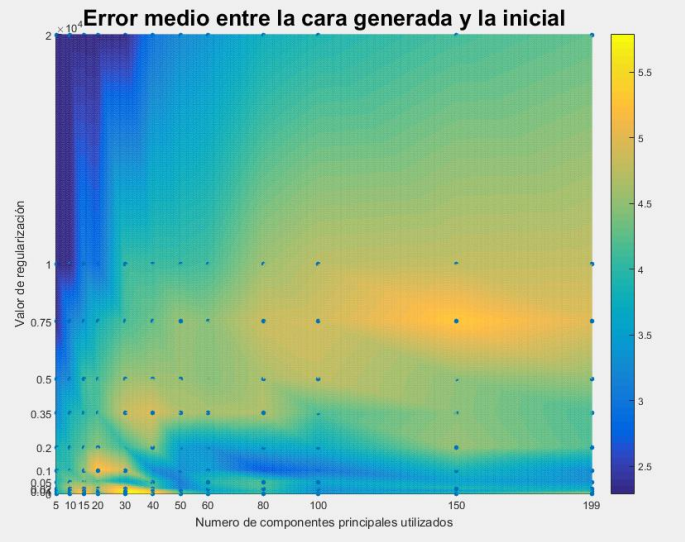

Fig. 6.28 *Error medio obtenido entre el modelo generado y el inicial para las distintas combinaciones de número de modos y valor de regularización*

Se observa que los resultados son parecidos a los obtenidos con Dasha; el error es menor con un mayor número de modos o con valores de regularización bajos. Los resultados visuales son también similares a los de Dasha. En la figura 6.29 se ve cómo; con la utilización de 199 modos, las altas componentes frecuenciales disminuyen al aumentar el valor de regularización.

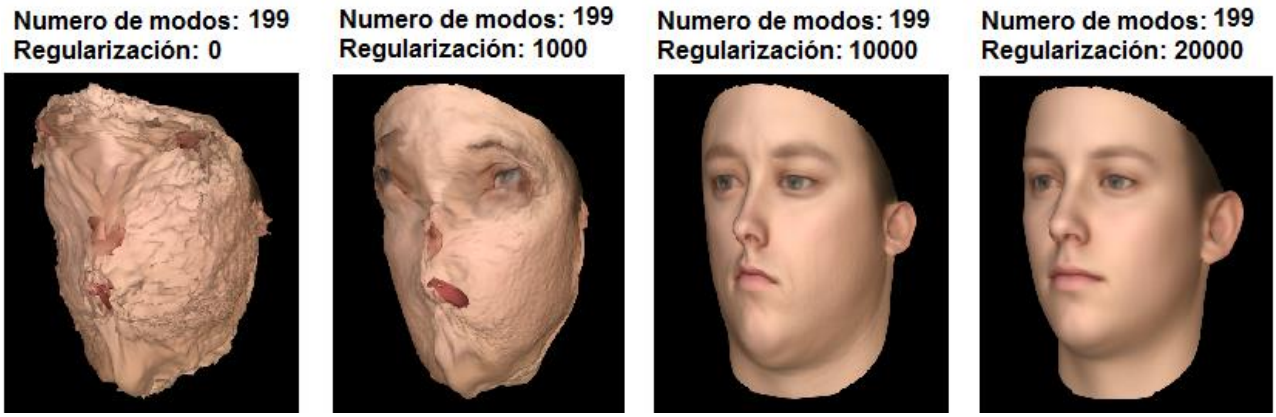

Fig. 6.29 *Consecuencia del uso de diferentes valores de regularización*

Dado el alto tiempo de simulación requerido para completar el experimento, se planteó el diezmar las nubes de puntos de Kinect y del modelo BFM mediante la función *pcdownsample*, que, aunque nos había proporcionado malos resultados en los experimentos con Dasha, pensamos que podría llegar a comportarse de una forma correcta para este caso. Sin embargo, los resultados no fueron distintos a los obtenidos en Dasha: de nuevo, el resultado fueron caras medias. Por ello, se consideró realizar la triangulación a la nube de puntos de Kinect para utilizar la función *MyCrustOpen*. Este es el motivo por el cual el programa *GetPointCloud* nos proporciona los índices de los tríos de puntos que forman un mismo triangulo.

Mientras se escribía el código para generar esta triangulación y realizar el experimento, se pensó que; sabiendo la relación que existe entre algunos puntos de la nube de Kinect y de la del BFM, se podría aplicar el algoritmo de ICP, que optimiza la combinación de componentes principales, únicamente a los puntos coincidentes y, de este modo, conseguir acelerar muchísimo el proceso. Así se hizo, por lo que en este caso no fue necesario el uso de la función *MyCrustOpen*. Los resultados obtenidos se muestran en la figura 6.30. También es posible medir el error existente únicamente en los puntos coincidentes. Este error de los puntos coincidentes, está representado en la figura 6.31.

En estas figuras, se observa que los errores medios calculados a partir de los errores de todos los puntos, disminuyen al aumentar el valor de regularización y al disminuir el número de modos utilizados; sin embargo, pasa lo contrario en el caso de los errores medios calculados a través de los puntos coincidentes. Esto nos demuestra que, al haber utilizado un algoritmo que determina la mejor combinación de componentes principales mediante, únicamente, los 24 puntos coincidentes entre ambas nubes de puntos, las caras generadas por este algoritmo presentaran valores de coincidencia mucho mayores en esos puntos mientras que los demás presentaran valores muy dispares respecto al modelo original. Para comprenderlo mejor, en la figura 6.32 podemos ver algunos ejemplos gráficos de las caras generadas.

Mediante estos ejemplo gráficos, vemos que las zonas en las que se encuentran los puntos coincidentes, permanecen más o menos constantes al aumentar el número de modos utilizados, sin embargo, las zonas en las que no tenemos coincidencia de puntos; como son la zona del cuello, de la mandíbula o de las orejas, sufren grandes variaciones que se reflejan en un aumento del error medio total. Por otro lado, también se observa que al aumentar los valores de regularización, las caras se van normalizando hasta llegar a la cara media, lo que supone un aumento del error medio de los puntos coincidentes y un descenso del error medio total; ya que las zonas que no han sido bien modeladas por el algoritmo, tienden a parecerse más a una cara real.

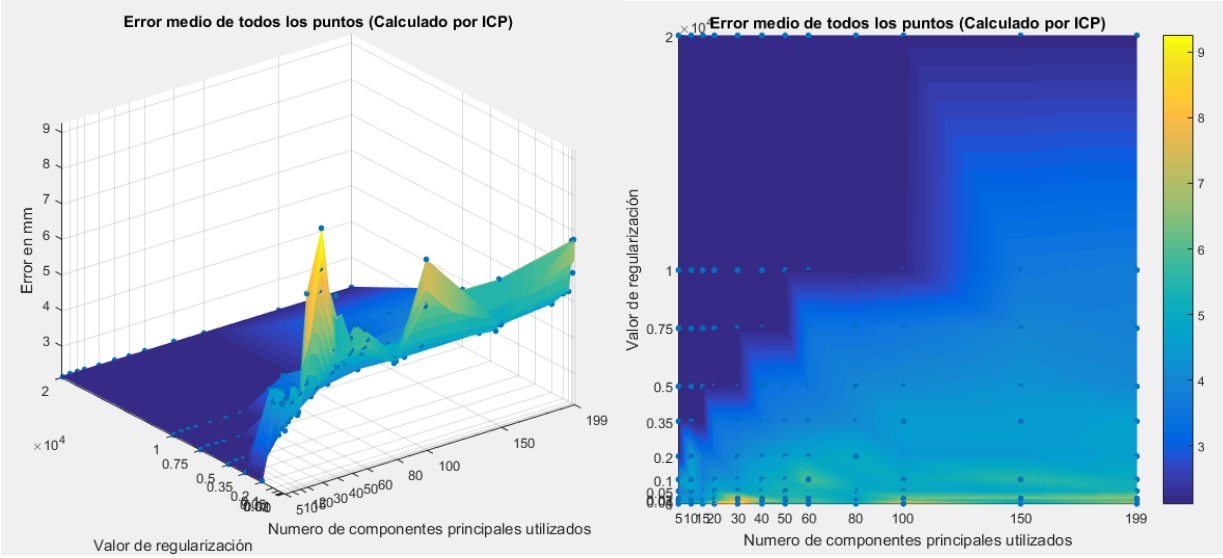

Fig. 6.30 *Errores medios calculados a partir de los errores de todos los puntos de las nubes de puntos*

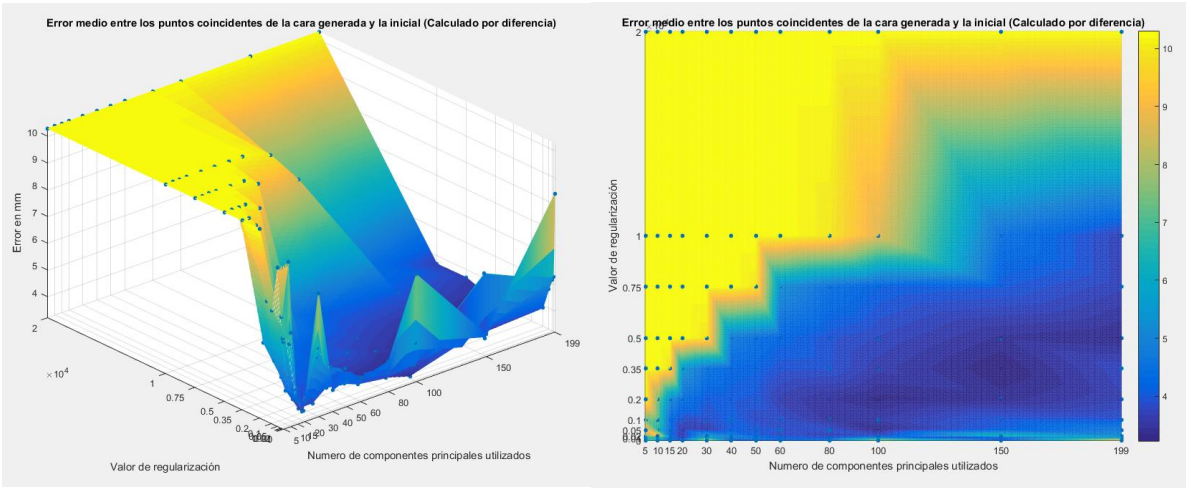

Fig. 6.31 *Errores medios calculados a partir de los puntos coincidentes de ambas nubes de puntos*

Fig. 6.32 *Modelos generados mediante el algoritmo de ICP*

A la vista de estos resultados y, para comprobar que se mantienen para distintos modelos de cara original, se pidió la colaboración de, esta vez, una mujer (la Dra. Arantzazu Villanueva), para obtener una nueva nube de puntos y someterla al mismo experimento. Los resultados de este nuevo experimento, realizado con un modelo de mujer, se muestran en las figuras 6.33 y 6.34.

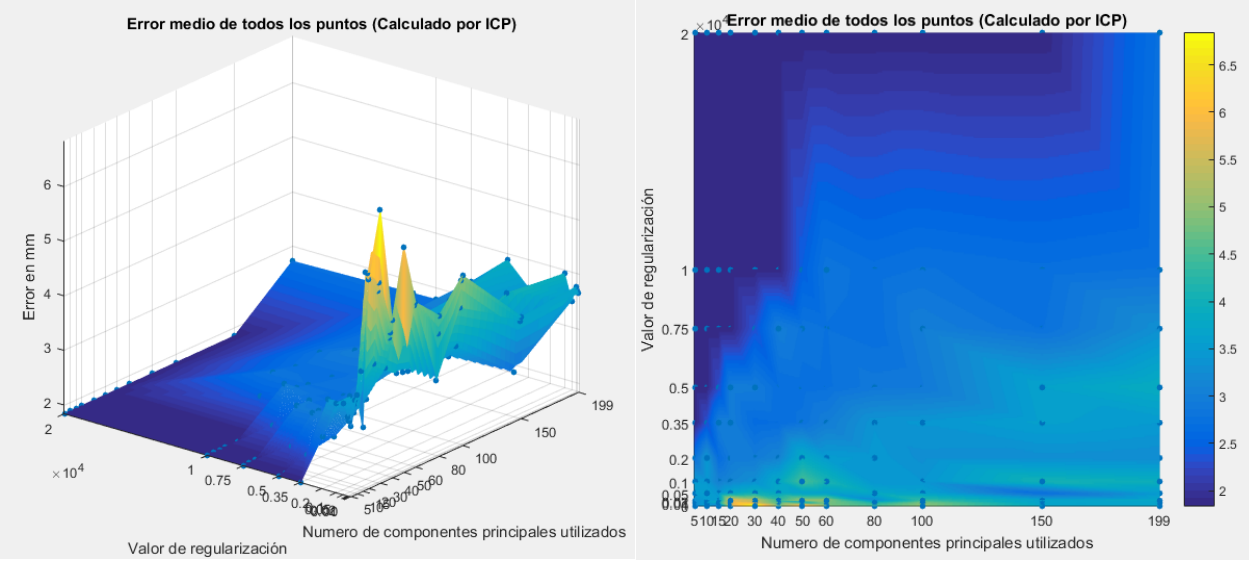

Fig. 6.33 *Errores medios calculados a partir de los errores de todos los puntos de las nubes de puntos para el caso del Arantzazu Villanueva*

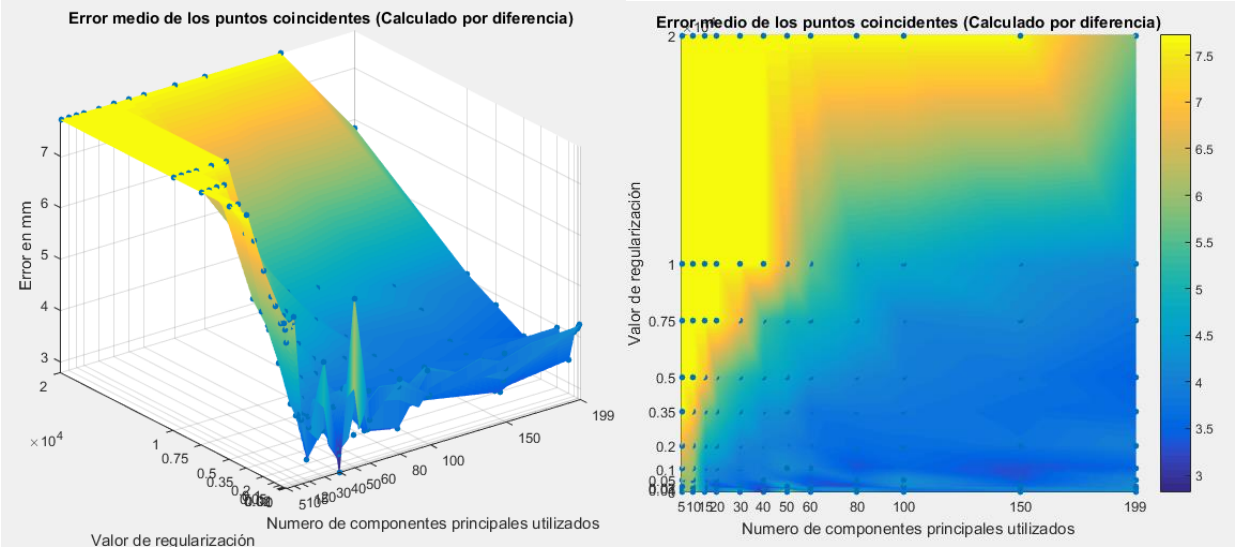

Fig. 6.34 *Errores medios calculados a partir de los puntos coincidentes de ambas nubes de puntos para el caso del Arantzazu Villanueva*

Existe una clarísima relación entre los resultados obtenidos en este experimento y los obtenidos anteriormente; las superficies de error se comportan de la misma forma y se encuentran en torno al mismo rango de error. En cuanto a las representaciones gráficas de las caras generadas sólo decir que también se comportan del mismo modo que en el caso anterior; tal y como se ve en la figura 6.35

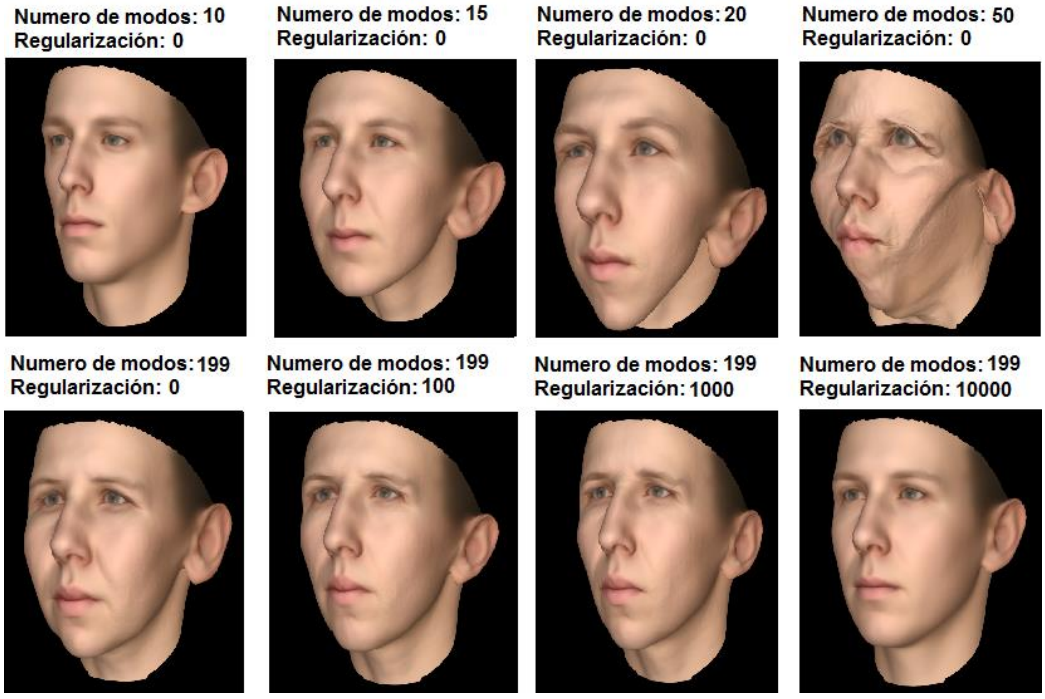

Fig. 6.35 *Modelos generados mediante el algoritmo de ICP para el caso de Arantzazu Villanueva*

Una vez realizados estos experimentos y sacado las conclusiones pertinentes, se pensó en la posibilidad de utilizar un algoritmo distinto al del ICP para obtener las combinaciones de componentes principales. En este caso se pensó en utilizar una simple resta entre pares de puntos coincidentes (4), es decir, a cada punto se le restó su coincidente y se calculó la media de todos esos errores.

$$E = \sum_{i=1}^{Nm} \|m_i - d_i\| \qquad (4)$$

Donde $\boldsymbol{Nm}$ tiene un valor de 24 (el total de los puntos coincidentes) y, al igual que en (3), $\boldsymbol{m_i}$ y $\boldsymbol{d_i}$ son los puntos en forma de vector (x,y,z) y aproximamos la norma como una suma del valor absoluto de las diferencias de cada componente vectorial.

Podemos ver los resultados de este proceso aplicado al primer modelo en las figuras 6.36 (error medio total), 6.37 (error medio de los puntos coincidentes) y 6.38 (representación gráfica).

Del mismo modo que con el uso del ICP, esta vez también realizamos otro experimento utilizando como modelo original el de la Dra. Arantzazu Villanueva. Podemos ver los resultados de este proceso en las figuras 6.39 (error medio total), 6.40 (error medio de los puntos coincidentes) y 6.41 (representación gráfica).

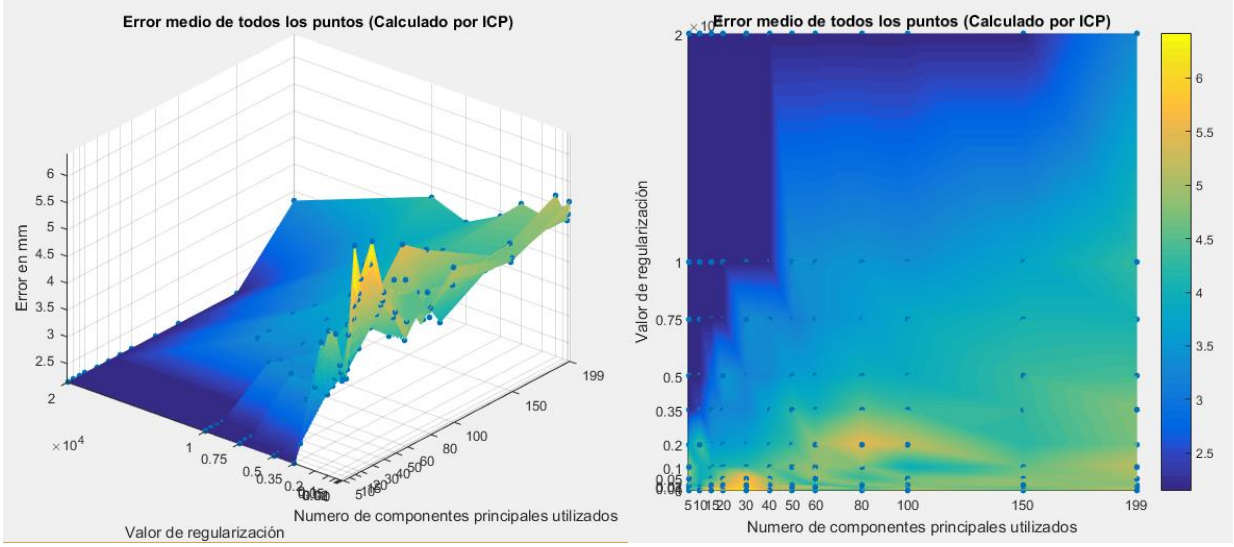

Fig. 6.36 *Errores medios calculados a partir de los errores de todos los puntos de las nubes de puntos mediante el algoritmo de diferencia*

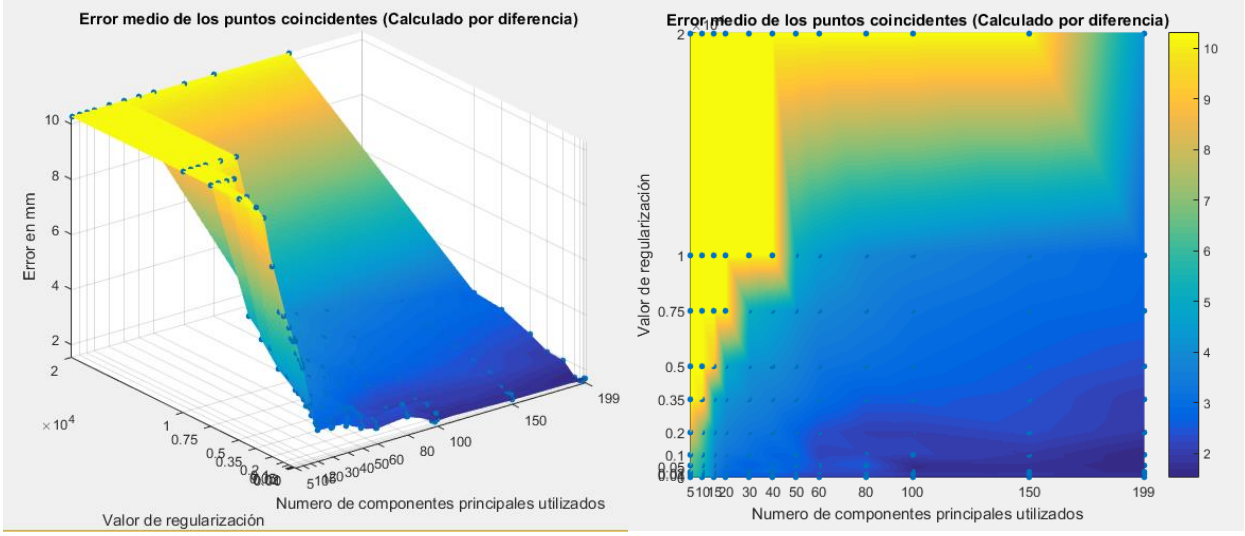

Fig. 6.37 *Errores medios calculados a partir de los puntos coincidentes de ambas nubes de puntos mediante el algoritmo de diferencia*

Fig. 6.38 *Modelos generados mediante el algoritmo de diferencia*

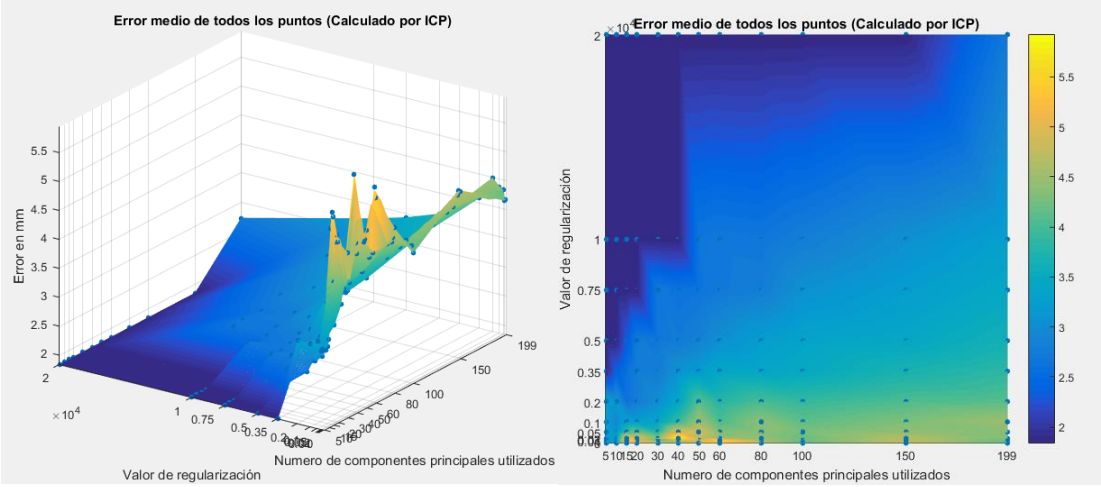

Fig. 6.39 *Errores medios calculados a partir de los errores de todos los puntos de las nubes de puntos mediante el algoritmo de diferencia para el caso de Arantzazu Villanueva*

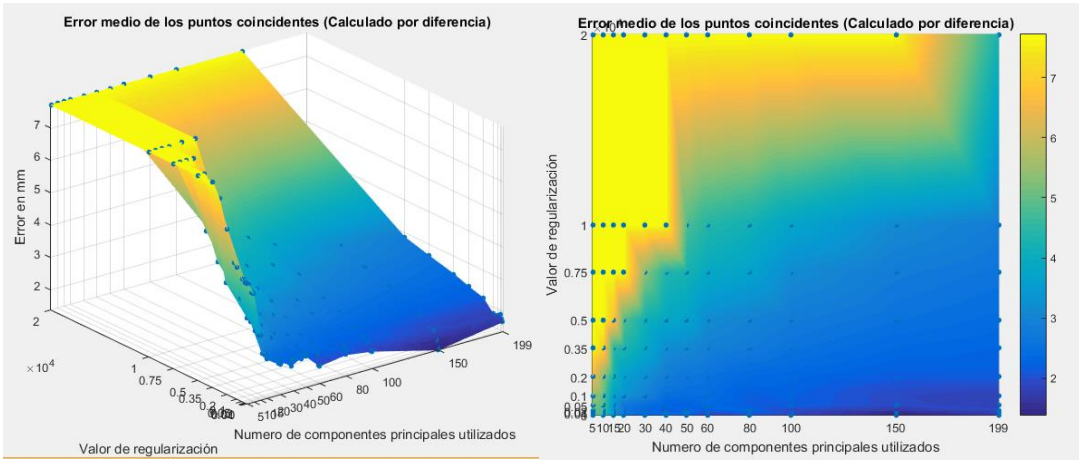

Fig. 6.40 *Errores medios calculados a partir de los puntos coincidentes de ambas nubes de puntos mediante el algoritmo de diferencia para el caso de Arantzazu Villanueva*

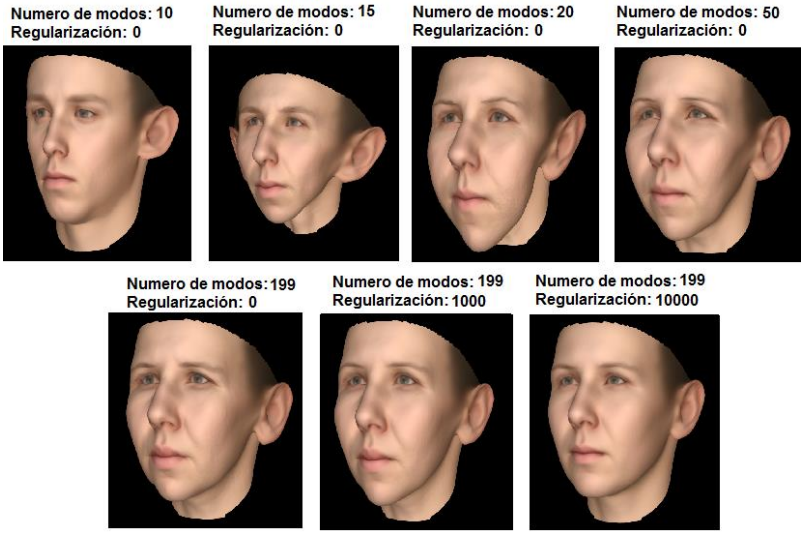

Fig. 6.41 *Modelos generados mediante el algoritmo de diferencia para el caso de Arantzazu Villanueva*

Vemos que, otra vez, ambos modelos se comportan de una forma muy similar. Si comparamos estos resultados con los obtenidos mediante el uso del algoritmo de ICP podemos ver como el algoritmo de la diferencia proporciona mejores valores de error medio total y de error medio de los puntos coincidentes.

Capítulo 7
Conclusiones y líneas futuras

Tras analizar los resultados, se puede llegar a la conclusión de que generar las combinaciones óptimas de los componentes principales mediante el uso exclusivo de los puntos coincidentes, proporciona unos resultados muy similares, en cuanto al rango de valores de error, a los obtenidos utilizando la totalidad de los puntos. Sin embargo, bajo mi punto de vista, creo que los resultados visuales son más fieles mediante el método de los 24 puntos coincidentes; además, los tiempos de simulación necesarios para realizar los experimentos resultan muchísimo menores con este método.

En cuanto al dilema de utilizar el algoritmo de ICP o el de la diferencia entre puntos coincidentes para la obtención de la combinación óptima de componentes principales, resulta difícil decantarse por uno de los dos. Las matemáticas nos dicen que el error mediante el método de la diferencia, proporciona mejores resultados y, a mi parecer (y esto sí que es totalmente subjetivo), creo que las caras generadas mediante este método se acercan más a las caras reales.

Como se ha comentado en varias ocasiones a lo largo del proyecto, existe una incoherencia entre el origen del sistema de coordenadas de los diferentes modelos utilizados, por lo que una clara línea de investigación sería la de conseguir realizar una unificación óptima de estos orígenes. Por otro lado, hemos visto que los errores medios de los puntos coincidentes pueden llegar a ser bastante pequeños (del orden de 1-2mm) por lo que si, de alguna forma obtuviésemos más pares de puntos coincidentes, los niveles de error total medio descenderían de forma significativa. Este descenso sería muy notable en el caso de utilizar un número elevado de modos, generando caras mucho más fieles al modelo original. Por ello, un claro objetivo de futuras investigaciones es el de intentar aumentar los puntos coincidentes entre ambas nubes de puntos. Otra línea de investigación podría ser la de obtener resultados más precisos acerca de la mejor combinación de valor de regularización y número de componentes principales.

Un punto importante en el que no hemos hecho hincapié en ningún apartado de este proyecto es que nosotros partimos de la base de que la nube de puntos proporcionada por Kinect v2, es un modelo fiable del sujeto. Sin embargo, no tenemos ningún dato acerca de lo bueno o malo que es este modelado. Así pues, sería muy

interesante realizar investigaciones que nos aportaran algo más de información acerca de este tema; ya que por mucho que nuestro algoritmo genere un modelado muy preciso del modelo obtenido a partir de Kinect, si este modelo no se ajusta a la realidad, nuestro trabajo no tendrá ningún sentido.

Para finalizar las conclusiones y cumplir con el objetivo principal de este proyecto, debemos valorar la utilidad de la herramienta Kinect v2 como método para la modelización 3D de la cabeza. El uso de este dispositivo para la generación de modelos de cabezas 3D podría estar perfectamente justificado si se completasen con éxito los objetivos declarados para futuras investigaciones pero, hoy por hoy, no tenemos suficiente control sobre él como para considerarlo una herramienta totalmente adecuada.

Capítulo 8
Manual de usuario

8.1 Obtención de la nube de puntos de Kinect

En este apartado detallaremos los pasos a seguir para obtener una nube de puntos mediante el programa *GetPointCloud*.

Lo primero de todo es abrir el proyecto *GetPointCloud*. Existen dos formas de hacerlo. La primera consiste en ir a la carpeta *GetPointCloud* y clicar sobre el archivo *GetPointCloud.sln* tal y como se muestra en la figura 8.1.

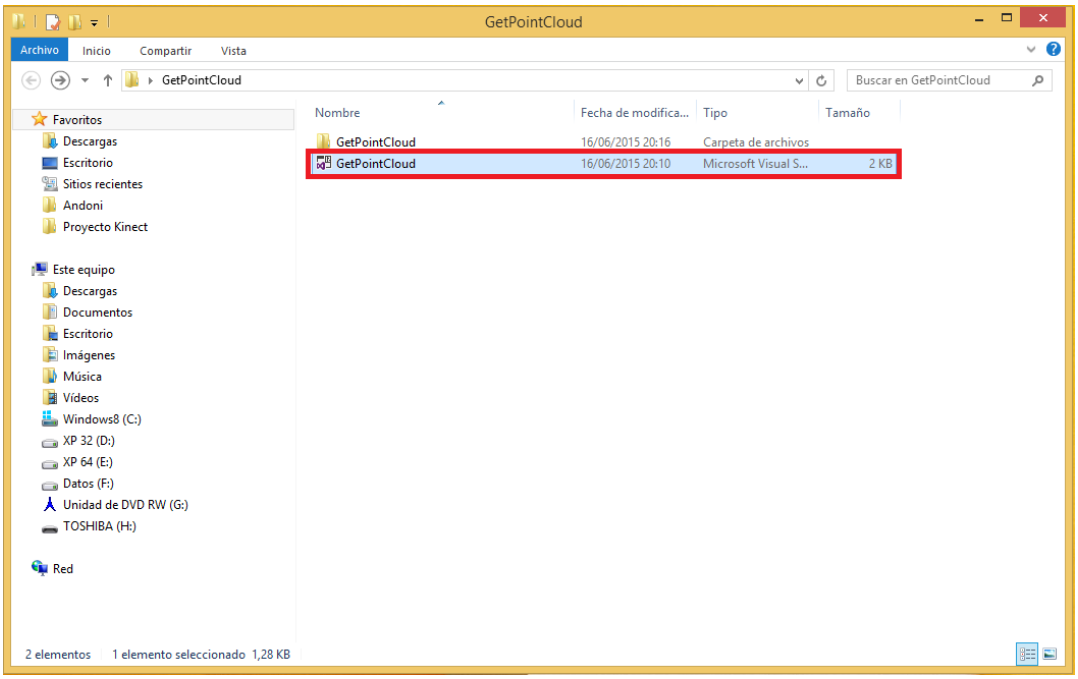

Fig. 8.1 *Clicar en el archivo GetPointCloud.sln*

La otra forma de abrir el proyecto es mediante el programa Visual Studio. Para ello haremos clic en *Archivo-->Abrir-->Proyecto o solución…* (ver figura 8.2) y seleccionaremos el archivo *GetPointCloud.sln*

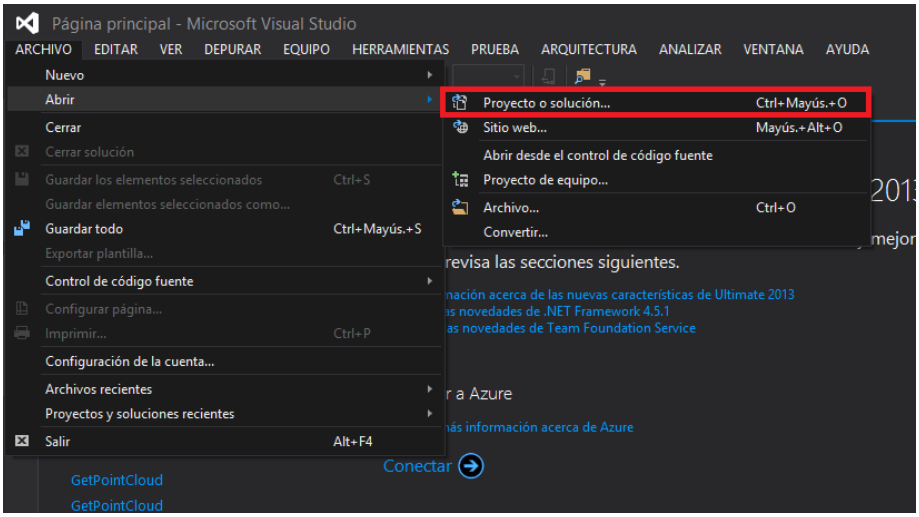

Fig. 8.2 *Archivo-->Abrir-->Proyecto o solución*

Una vez abierto el proyecto, haremos doble clic en la pestaña *MainWindow.xaml.cs* dentro del Explorador de soluciones (ver figura 8.3). Esto nos generará una pestaña para poder ver el código del programa.

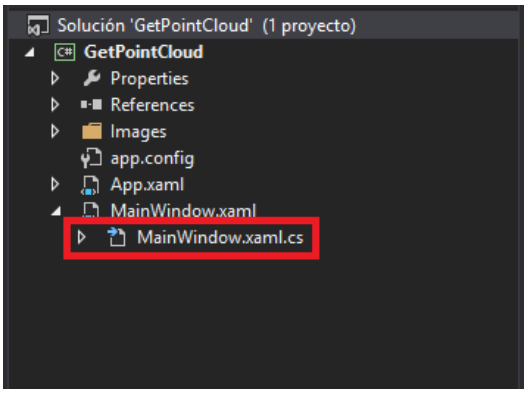

Fig. 8.3 *Doble clic--> MainWindow.xaml.cs*

Ahora pulsaremos *Ctrl+F* para abrir el buscador y escribiremos ***DIRECTORIO*** tal y como se muestra en la figura 8.4.

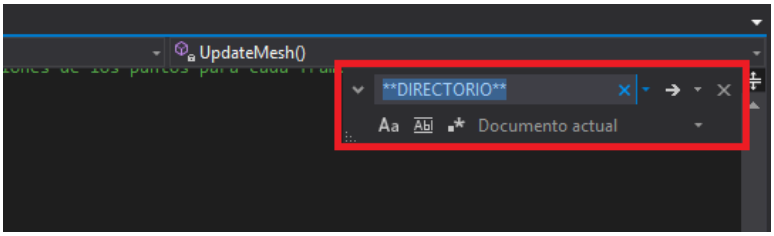

Fig. 8.4 *Ctrl+F--> **DIRECTORIO***

El programa buscará este conjunto de caracteres en el proyecto y nos conducirá hasta la línea de código en la que está escrito (ver figura 8.5). Debajo de esta línea de código está el directorio en el cual vamos a guardar los datos generados por el programa así que modificaremos la ruta para guardar los datos en la carpeta que deseemos.

```
//**DIRECTORIO**
string coreFolder = @"C:\Users\alarumbe\Desktop\Andoni\Datos";
```

Fig. 8.5 *Línea de código en la que está escrito **DIRECTORIO***

Una vez configurado el directorio en el que se guardaran nuestros datos, ejecutaremos el programa clicando en el botón *Iniciar* situado debajo de la Barra de Herramientas (figura 8.6).

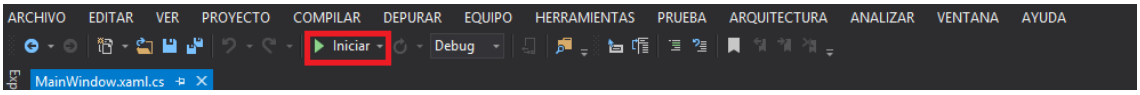

Fig. 8.6 *Clic en el botón Iniciar*

Nuestro programa ya estará ejecutándose. Ahora, tendremos que pulsar *Start Capture* y, como se ha visto en el capítulo 6, proporcionar datos de nuestra cara mostrando los perfiles, una vista frontal y una vista frontal inclinada (es posible que tengamos que alejarnos ligeramente del dispositivo Kinect o incluso levantarnos para que nos detecte). En la figura 8.7 podemos ver un ejemplo de las vistas que Kinect necesita para generar el modelo.

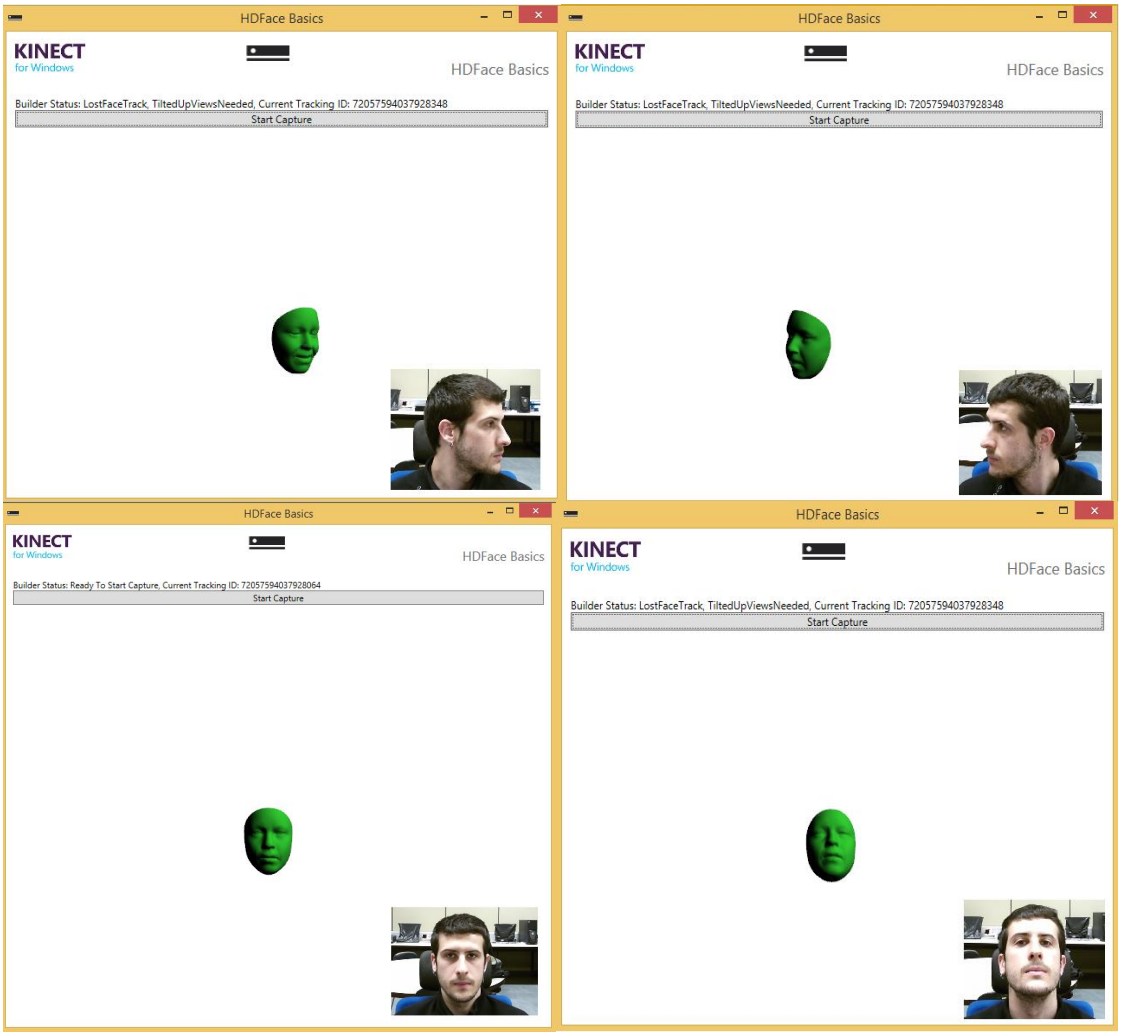

Fig. 8.7 *Vistas necesarias para generar el modelo*

Una vez Kinect haya generado el modelo, automáticamente se creará una nueva carpeta en el ruta que hemos determinado anteriormente. El nombre de la carpeta que contendrá los datos obtenidos será por defecto "*1*", si ya existe una carpeta con datos que se llame del mismo modo, el nombre de la carpeta generada será "*2*" y así sucesivamente.

Por último, nos queda cambiar el formato de los datos obtenidos de Kinect ya que, por defecto, nuestra aplicación guarda los espacios como ";" y los puntos de los decimales como ",". Para ello abriremos con un editor de texto el archivo *points.txt* de la carpeta generada por la aplicación tal y como muestra la figura 8.8.

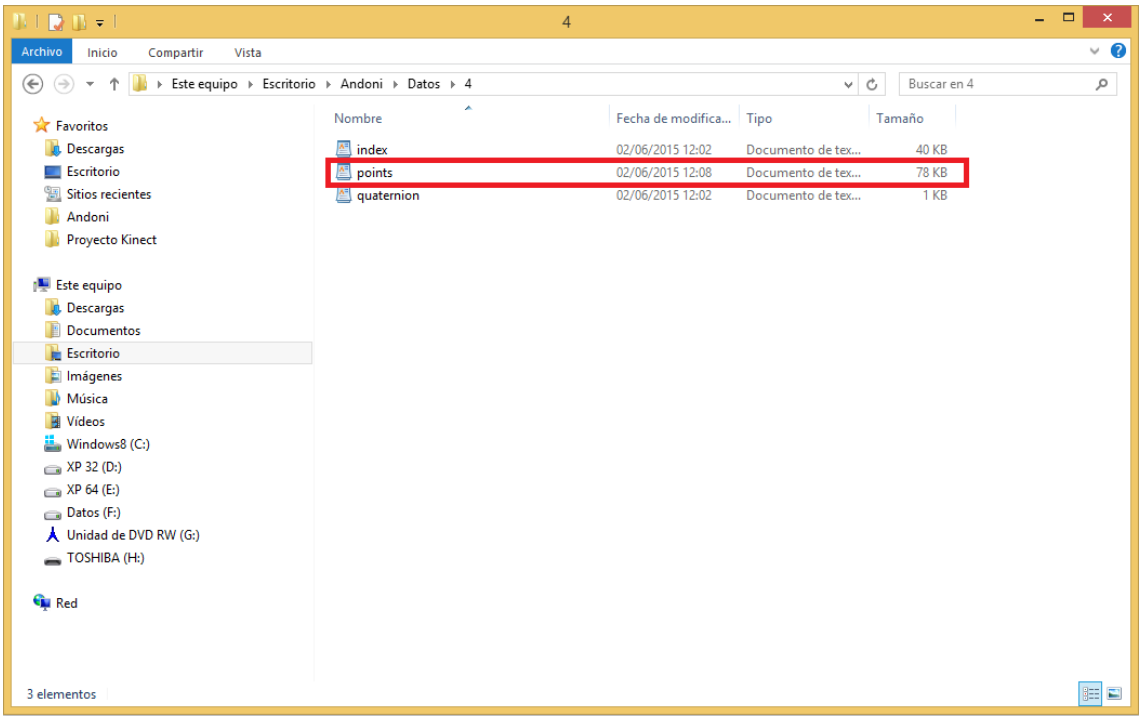

Fig. 8.8 *Abrir el archivo points.txt*

Mediante la herramienta *Reemplazar* cambiaremos las comas (",") por puntos (".") y, una vez realizado este cambio, los punto y coma (";") por comas (","). **Es muy importante respetar el orden de los reemplazos.**

Si se quieren utilizar los demás archivos para trabajar con ellos en MATLAB, se deberá realizar un procedimiento similar.

Una vez realizados estos pasos, ya tendremos la nube de puntos lista para ser utilizada en MATLAB mediante un simple "Copia-Pega".

8.2 Programa "*showFaces*"

En este apartado hablaremos sobre el programa "***showFaces***" y detallaremos los pasos a seguir para generar una cara a partir del mismo.

8.2.1 Introducción

El programa ***showFaces*** es una interfaz gráfica creada durante el desarrollo del proyecto para facilitar el visionado de las caras generadas en nuestros experimentos. Es un programa muy simple que nos permite cargar un archivo de extensión ".mat" en el cual estén guardados los 12x11 componentes principales resultantes de un experimento (11 valores de regularización para cada uno de los números de modos utilizados) y generar cualquiera de las caras resultantes de aplicar esos componentes principales. El entorno del programa se muestra en la figura 8.9.

Fig. 8.9 *Entorno del programa showFaces*

Como se observa en la imagen, este entorno está compuesto por 5 espacios:

- **El espacio 1** nos muestra el fichero .mat del cual estamos extrayendo los componentes principales para generar las caras.
- **El espacio 2** nos permite elegir el número de modos y el valor de regularización que queremos
- Al pulsar en el botón "*Create*" **del espacio 3** se generará la cara con los valores seleccionados en el espacio 2.
- **El espacio 4** nos permite introducir o cambiar el fichero .mat del cual extraeremos los datos
- Al pulsar en el botón "*Cargar fichero*" **del espacio 5** se cargará el fichero introducido en el espacio 4.

8.2.2 Manual de usuario

El primer paso al ejecutar nuestro programa será introducir, a través del cuadro de texto, el fichero del cual queremos extraer los componentes principales. Para ello clicaremos encima del cuadro de texto situado en el espacio 4 (ver figura 8.9) y escribiremos el nombre del fichero que queramos cargar (**Importante:** la extensión .mat debe **estar incluida** en el nombre del fichero). Tras haber escrito el nombre del fichero que queremos cargar pulsaremos en *Cargar fichero*. Debemos visualizar el nombre del fichero que hemos elegido en el espacio 1 tal y como se ve en la figura 8.10.

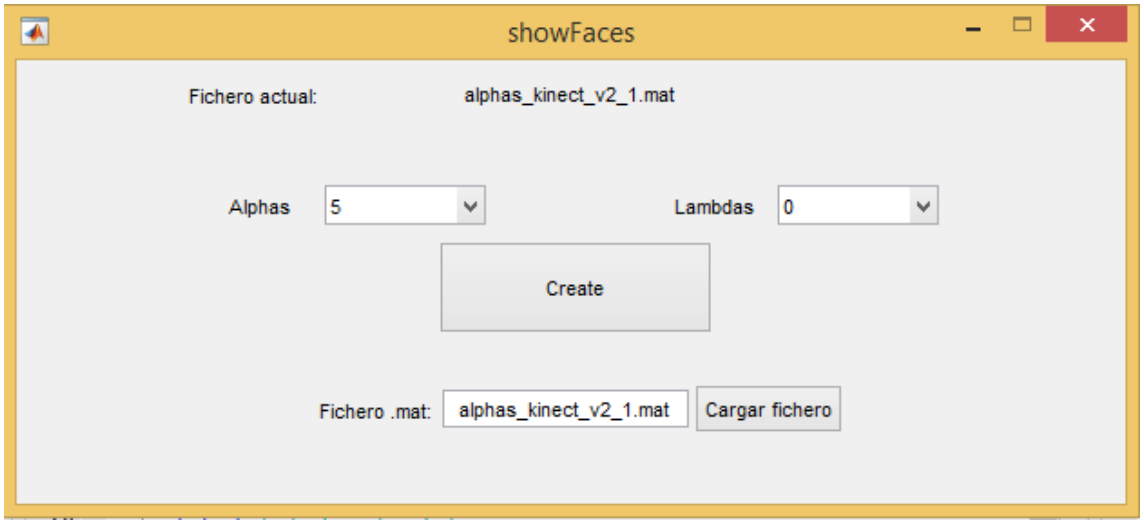

Fig. 8.10 *Debemos ver el fichero seleccionado donde pone Fichero actual.*

Una vez cargado nuestro fichero, tendremos que seleccionar la combinación de número de modos (alphas) y de valor de regularización (lambdas) que queramos usar para generar nuestra cara. Para ello expandiremos los menús desplegables del espacio 2 y seleccionaremos los valores que deseemos. En la figura 8.11 podemos ver estos menús desplegados.

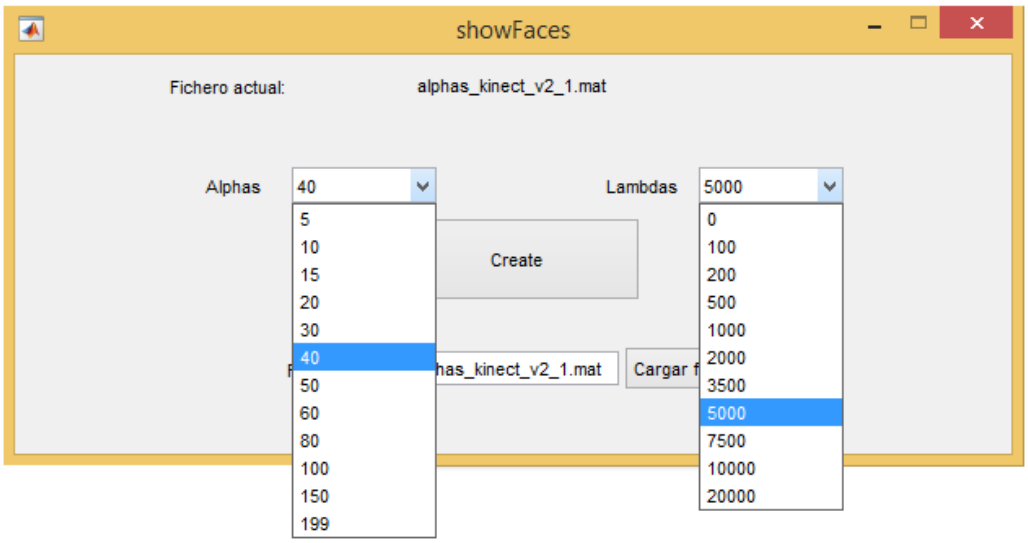

Fig. 8.11 *Menus desplegables de alpha y lambda*

Cuando ya tengamos los valores de alpha y lambda seleccionados, pulsaremos en el botón *Create,* obteniendo la cara. Cabe destacar que si pulsamos el botón *Create* sin tener cargado ningún fichero, el programa nos mostrara por pantalla el siguiente mensaje: "*Por favor, cargue un archivo .mat*".

En la figura 8.12 se muestra el resultado final tras cargar un fichero y seleccionar los valores de alpha y lambda .

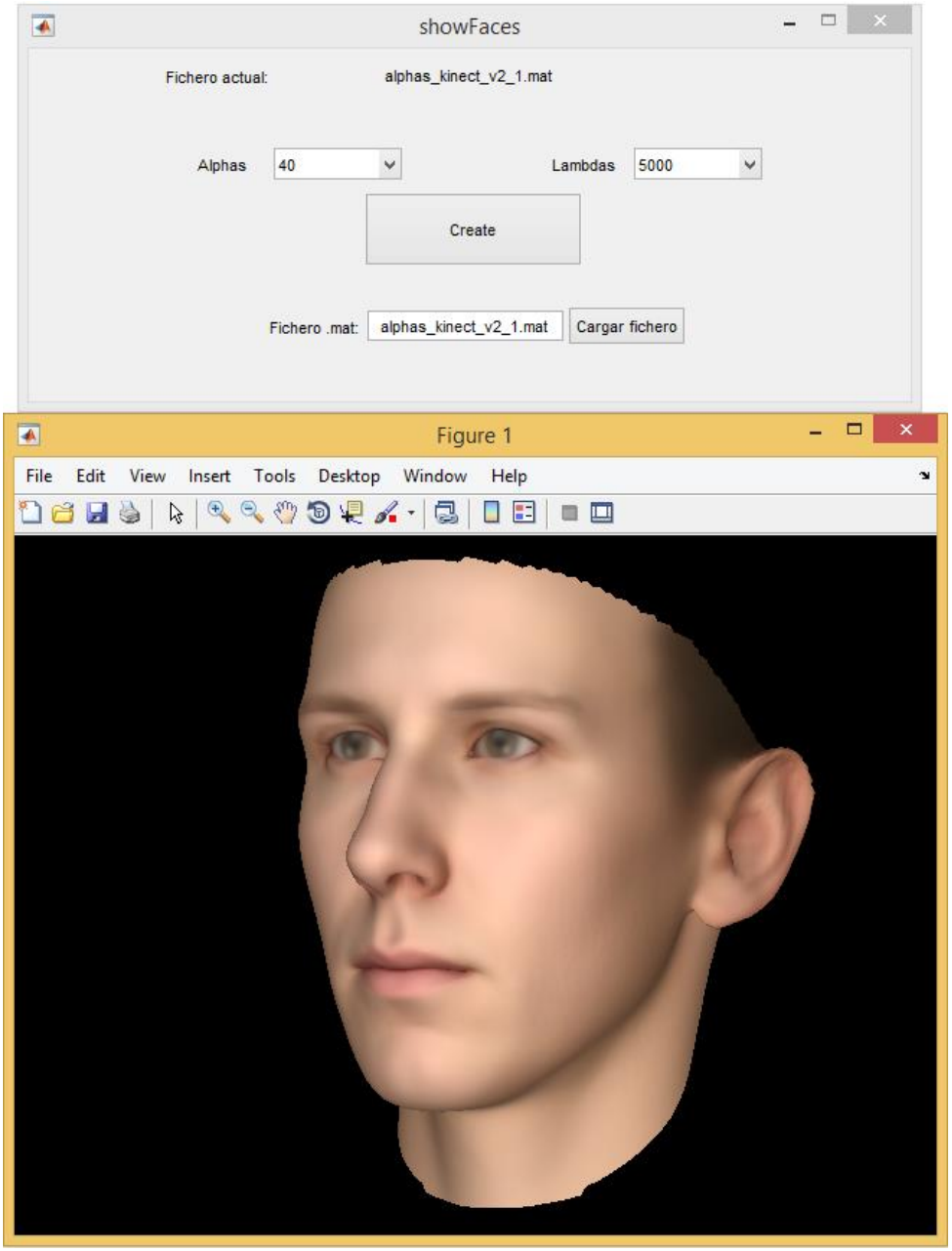

Fig. 8.12 *Visualización de una cara mediante el programa showFaces*

Bibliografía

[1] Murphy-Chutorian, E. y M. Trivedi, M. (2008). Head Pose Estimation in Computer Vision: A Survey [Figura]. Recuperado de: http://people.ict.usc.edu/~gratch/CSCI534/Head%20Pose%20estimation.pdf

[2] Lanier, J. (2013). Circling, Squaring, and Triangulating [Figura]. Recuperado de: http://mathmunch.org/2013/05/08/circling-squaring-and-triangulating/

[3] C. Tomasi y T. Kanade, Shape and motion from image streams under orthography: a factorization method, *International Journal of Computer Vision* (9) (1992) 137–154.

[4] Gonzalez-Mora1, J., De la Torre, F., Guil, N. y L. Zapata, E. (2010). Learning a generic 3D face model from 2D image databases using incremental structure from motion, *Elsevier*.

[5] Lower, B., Relyea, R., Kaplan, J., Schwesinger, M., Sirignano, C., Simari, M., Marien, J. y Meekhof, C. (2013). Programming Kinect for Windows v2 Jump Start [Figuras] Recuperado de: http://www.microsoftvirtualacademy.com/training-courses/programming-kinect-for-windows-v2-jump-start

[6] Paysan, P., Knothe, R., Amberg, B., Romdhani, S. y Vetter, T. (2009). A 3D Face Model for Pose and Illumination Invariant Face Recognition [Figuras]. Recuperado de: http://gravis.cs.unibas.ch/publications/2009/BFModel09.pdf

[7] Gallardo, D. (2012). Alineamiento 3D a partir de nubes de puntos [Ecuación]. Recuperado de: http://www.dccia.ua.es/dccia/inf/asignaturas/Vision/vision-tema6.pdf

[8] S. Pandzic, I. y Forchheimer, R. (2002). MPEG-4 Facial Animation: The standard, implementations and applications, [Figura]. Recuperado de: http://www.visagetechnologies.com/uploads/2012/08/MPEG-4FBAOverview.pdf